1   DATA RELATIONSHIPS PROCESSOR

2   WITH UNLIMITED EXPANSION CAPABILITY

3

4   Karol Doktor

5

6

7 BACKGROUND OF THE INVENTION

8 1.   Cross Reference to Microfiche Appendix

9   This application includes a plurality of computer

10 program listings (modules) in the form of a Microfiche

11 Appendix which is being filed concurrently herewith as 1162

12 frames (not counting target and title frames) distributed

13 over 20 sheets of microfiche in accordance with 37 C.F.R. §

14 1.96. The disclosed computer program listings are

15 incorporated into this specification by reference but it

16 should be noted that the source code and/or the resultant

17 object code of the disclosed program modules are subject to

18 copyright protection. The copyright owner has no objection

19 to the facsimile reproduction by anyone of the patent

20 document (or the patent disclosure as it appears in the

21 files or records of the U.S. Patent and Trademark Office)

22 for the sole purpose of studying the disclosure but

23 otherwise reserves all other rights to the disclosed

24 computer program modules including the right to reproduce

25 said computer program modules in machine-executable form.

26

27 2.   Field of Invention

28   The present invention relates generally to computer

29 database management systems and more specifically to

30 apparatus and methods for modifying and searching through

31 large scale databases at high speed.

32

33 3.   Description of Related Art

34   Modern computer systems are capable of storing

35 voluminous amounts of information in bulk storage means such

36 as magnetic disk banks. The volume of stored information

37 can be many times that of the textual information stored in

38 a conventional encyclopedia or in the telephone directory of

1 a large city. Moreover, modern computer systems can sift
2 through the contents of their bulk storage means at
3 extremely high speed, accessing as many as one million bytes
4 of information or more per second (a byte is a string of
5 eight bits, equivalent to approximately one character of
6 text in layman's terms). Despite this capability, it may
7 take an undesirably long time (i.e., hours or days) to
8 retrieve desired pieces of information. In commercial
9 settings such as financial data storage facilities, there
10 will be literally billions of pieces of information that
11 could be sifted through before the right one or more pieces
12 of information are found. Thus, even at speeds of one
13 million examinations per second, it can take thousands of
14 seconds (many hours) to retrieve a desired piece of informa-
15 tion. Efficient organization of the stored information is
16 needed in order to minimize retrieval time.
17   The methods by which pieces of information are
18 organized within a computer, searched through or
19 reorganized, often parallel techniques used by older types
20 of manual information processing systems. A well known
21 example of a manual system is the index card catalog found
22 in public libraries. Such a card catalog consists of a
23 large number of uniformly dimensioned paper cards which are
24 serially stacked in one or more trays. The cards are
25 physically positioned such that each card is directly
26 adjacent to no more than two others (for each typical
27 examination there is a preceding card, the card under
28 examination and a following card in the stack). On the
29 front surface of each index card a librarian enters, in left
30 to right sequence; the last name of an author, the first
31 name of the author, the title of a single book which the
32 author wrote and a shelf number indicating the physical
33 location within the library where the one book may be
34 found. Each of these four entries may be referred to as a
35 "column" entry. Sufficient surface area must be available
36 on each card to contain the largest of conceivable entries.
37   After the entries are made, the index cards are stacked
38 one after the next in alphabetical order, according to the

1 author's last name and then according to the author's first
2 name and then by title. This defines a "key-sequenced" type
3 of database whose primary sort key is the author's name.
4 The examination position of each card is defined relative to
5 the contents of preceding and following cards in the
6 stack. That is, when cards are examined, each intermediate
7 card is examined immediately after its alphabetically
8 preceding card and immediately before its alphabetically
9 succeeding card. When a new book is acquired, the key-
10 sequenced database is easily "updated" by inserting a new
11 card between two previously created cards. Similarly, if a
12 book is removed from the collection, its card is simply
13 pulled from the card stack to reflect the change.
14   If a library user has an inquiry respecting the
15 location of a particular book or the titles of several books
16 written by a named author, the librarian may quickly search
17 through the alphabetically ordered set of index cards and
18 retrieve the requested information. However, if a library
19 user has an inquiry which is not keyed to an author's name,
20 the search and retrieval process can require substantially
21 more time; the worst case scenario being that for each
22 inquiry the librarian has to physically sift through and
23 examine each card in the entire catalog. As an example of
24 such a scenario, suppose that an inquiring reader asks for
25 all books in the library where the author's first name is
26 John and the title of the book contains the word "neighbor"
27 or a synonym thereof. Although it is conceptually possible
28 to answer this inquiry using the information within the
29 catalog, the time for such a search may be impractically
30 long, and hence, while the information is theoretically
31 available, it is not realistically accessible.
32   To handle the more common types of inquiries, libraries
33 often keep redundant sets of index cards. One set of cards
34 is sorted according to author names and another set is
35 sorted according to the subject matter of each book. This
36 form of redundant storage is disadvantageous because the
37 size of the card catalog is doubled and hence, the cost of
38 information storage is doubled. Also, because two index

1 cards must be generated for each new book added to the

2 collection the cost of updating the catalog is also doubled.

3　　　The size of a library collection tends to grow over

4 time as more and more books are acquired. During the same

5 time, more and more index cards are added to the catalog.

6 The resulting stack of cards, which may be viewed as a kind

7 of "database", therefore grows both in size and in worth.

8 The "worth" of the card-based system may be defined in part

9 as the accumulated cost of all work that is expended in

10 creating each new index card and in inserting the card into

11 an appropriate spot in the stack.

12　　　As time goes by, not only does the worth and size of

13 the database grow, but new technologies, new rules, new

14 services, etc., begin to emerge and the information

15 requirements placed on the system change. Some of these

16 changes may call for a radical reorganization of the card

17 catalog system. In such cases, a great deal of work

18 previously expended to create the catalog system may have to

19 be discarded and replaced with new work.

20　　　For the sake of example, let it be supposed that the

21 library acquires a new microfilm machine which stores copies

22 of a large number of autobiographies. The autobiographies

23 discuss the life and literary works of many authors whose

24 books are kept in the library. Let it further be supposed

25 that the original, first card catalog system is now required

26 to cross reference each book to the microfilm location (or

27 plural locations) of its author's (or plural authors')

28 autobiographies. In such a case, the card catalog system

29 needs to be modified by adding at least one additional

30 column of information to each index card to indicate the

31 microfilm storage locations of the relevant one or more

32 autobiographies.

33　　　We will assume here that there is not enough surface

34 area available on the current index cards for adding the new

35 information. Larger cards are therefore purchased, the

36 information from the old cards is copied to the new cards,

37 and finally, the new microfilm cross referencing information

38 is added to the larger cards. This type of activity will be

1 referred to here as "restructuring" the database.

2 Now let us suppose, that as more time goes by, an
3 additional but previously unanticipated, cross indexing
4 category is required because of the introduction of a newer
5 technology or a new government regulation. It might be that
6 the just revised and enlarged second card system does not
7 have the capacity to handle the demands of the newer
8 technology or regulation. In such a situation, a third card
9 system has to be constructed from scratch. The value of
10 work put into the creation of the just-revised second system
11 is lost. As more time passes and further changes emerge in
12 technology, regulations, etc., it is possible that more
13 major organizational changes will have to be made to the
14 catalog system. Time after time, a system will be built up
15 only to be later scrapped because it fails to anticipate a
16 new type of information storage and retrieval operation.
17 This is quite wasteful.

18 Although computerized database systems are in many ways
19 different from manual systems, the computerized information
20 storage and retrieval systems of the prior art are analogous
21 to manual systems in that the computerized databases require
22 similar restructuring every time a new category of
23 information relationships or a new type of inquiry is
24 created.

25 At a fundamental level, separate pieces of information
26 are stored within a computerized database system as a large
27 number of relatively short strings of binary bits where each
28 string has finite length. The bit strings are distributed
29 spacially within a tangible medium of data storage such as
30 an array of magnetic disks, optical devices or other
31 information representing means capable of providing mass
32 storage. Each bit is represented by a magnetic flux
33 reversal, an optical perturbation and/or some other variance
34 in the physical attributes of a data storage medium. A
35 transducer or amplifier means converts these variances into
36 signals (e.g., electrical, magnetic, or optical) which can
37 be processed on a digital data processing machine. Each
38 string of bits is often uniquely identified by its physical
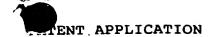
1 location or by a logical storage address. Some bit strings
2 may function as address pointers, rather than as the final
3 pieces of "real" information which a database user wishes to
4 obtain. The address pointers are used to create so-called
5 "threaded list" organizations of data wherein logical links
6 between a first informational "object" (first piece of real
7 data) and a second informational "object" (second piece of
8 real data) are established by a chain of direct or indirect
9 address pointers. The user-desired objects of real
10 information themselves can be represented by a collection of
11 one or more physically or logically connected strings.
12   Typically, "tables" of information are created within
13 the mass storage means of the computerized system. A
14 horizontal "row" of related objects, which is analogous to a
15 single card in a card catalog system, may be defined by
16 placing the corresponding bit strings of the objects in
17 physical or address proximity with each other. Logical
18 interconnections may be defined between different rows by
19 using ancillary pointers (which are not considered here as
20 the "real" data sought by a database user). A serial
21 sequence of "rows" (analogous to a stack of cards) is then
22 defined by linking one row to another according to a
23 predefined sorting algorithm using threaded list techniques.
24   A vast number of different linking "threads" may be
25 defined in this way through a database table having millions
26 or billions of binary information bits. Unlike manual
27 systems, the same collection of rows (which replaces the
28 manual stack of cards) can be simultaneously ordered in many
29 different ways by utilizing a multiplicity of threaded paths
30 so that redundant data storage is not necessary. Searches
31 and updates may be performed by following a prespecified
32 thread from one row to the next until a sought piece of
33 information (or its address) is found within a table. A
34 threaded-list type of table can be "updated" in a manner
35 similar to manual card systems by breaking open a logical
36 thread within the list, at a desired point, and inserting a
37 new row (card) or removing an obsolete row at the opened
38 spot.

1   Tables are often constructed according to a "key-
2 sequenced" approach. One column of a threaded-list table is
3 designated as the sort-key column and the entries in that
4 column are designated as "sort keys". Address pointers are
5 used to link one row of the table to another row according
6 to a predefined sequencing algorithm which orders the
7 entries (sort-keys) of the sort column as desired (i.e.,
8 alphabetically, numerically or otherwise). Once a table is
9 so sorted according to the entries of its sort column, it
10 becomes a simple task to search down the sort column looking
11 for an alphabetically, numerically or otherwise ordered
12 piece of data. Other pieces of data which are located
13 within the row of each sort key can then be examined in the
14 same sequence that each sort key is examined. Any column
15 can serve as the sort column and its entries as the sort
16 keys. Thus a table having a large plurality of columns can
17 be sorted according to a large number of sorting algorithms.
18   The key-sequencing method gives tremendous flexibility
19 to a computerized database but not without a price. Each
20 access to the memory location of a list-threading address
21 pointer or to the memory location of a sort-key or to the
22 memory area of "real" data which is located adjacent to a
23 sort-key takes time. As more and more accesses are required
24 to fetch pointers and keys leading to the memory location of
25 a piece of sought-after information ("real data"), the
26 response time to an inquiry increases and system performance
27 suffers.
28   There is certain class of computerized databases which
29 are referred to as "relational databases". Such database
30 systems normally use threaded list techniques to define a
31 plurality of key-sequenced "tables". Each table contains at
32 least two columns. One column serves as the sort column
33 while a second or further columns of the table store either
34 the real data that is being sought or additional sort-key
35 data which will ultimately lead to a sought-after piece of
36 real data. The rows of the table are examined in an ordered
37 fashion according to the contents of the sort column.
38 Target data is located by first threading down the sort

1 column and thus moving through the chain of rows within a

2 table according to a prespecified sort algorithm until a

3 specific sort-key is found. Then the corresponding row is

4 examined horizontally and the target data (real data or the

5 next key) is extracted from that row.

6   An example of "real" data would be the full-legal names

7 of unique persons such as in the character strings,

8 "Mr. Harry W. Jones", "Mrs. Barbara R. Smith", etc. The

9 sort-key can be a number which is stored adjacent to the

10 full name and which sequences the names (real data)

11 according to any of a wide variety of ordering patterns

12 including by age, by height, by residential address,

13 alphabetically, etc. Because the real data (e.g., full name

14 of a person) is stored in a separate column, it is

15 independent from the sort key data. A large variety of

16 different relations can therefore be established between a

17 first piece of real data (e.g., a first person's name) and a

18 second piece of real data (e.g., a second person's name)

19 simply by changing the sort keys that are stored in the

20 separate sort column (e.g., who is older than whom, who is

21 taller, etc.). Plural orderings of the real data can be

22 obtained at one time by providing many columns in one table,

23 by storing alternate keys in the columns and by choosing one

24 or more of these columns as the primary sort key column.

25   Relational database systems often include tables that

26 do not store real data in a column adjacent to their

27 sort-key column, but rather store a secondary key number

28 which directs a searcher to a row in another key-sequenced

29 table where a matching key number is held together with

30 either a piece of sought-after real data or yet another

31 forward referencing key number (e.g., an entry which in

32 effect says "find the row which holds key number x of yet

33 another table for further details"). With this indirect

34 key-sequenced approach, a large number of tables can be

35 simultaneously updated by changing one entry in a "base"

36 table.

37   Relational database tables are normally organized to

38 create implied set and subset "relations" between their

1 respective items of pre-stored information. The elements of
2 the lowest level subsets are stored in base tables and
3 higher level sets are built by defining, in other tables,
4 combinations of keys which point to the base tables. · The
5 implied relations between elements cannot be discerned by
6 simply inspecting the raw data of each table. Instead,
7 relations are flushed out only with the aid of an access
8 control program which determines in its randomly-distributed
9 object code, which table to examine first and what column to
10 look at before beginning to search down the table's column
11 for a key number and, when that key number is found, what
12 other column to look at for the real data or a next key
13 number. Relations between various "entities" of a
14 relational database are implied by the sequence in which the
15 computer accesses them.
16   By way of a concrete example, consider a first
17 relational table (Names-Table) which lists the names of a
18 large number of people in telephone directory style. Each
19 name (each separate item of real data) is paired to a unique
20 key number and the rows of this Names-Table are sorted
21 sequentially according to the key number. A second
22 relational table may be provided in the database
23 (Cars-Table) which lists automobile (vehicle) identification
24 numbers (VIN) each paired in its row with a second key
25 number. If the second key number is matched by a
26 corresponding key number in the first table, then a
27 relationship might be implied between the entries of the two
28 separate tables (Names-Table and Cars-Table). The "implied"
29 relationship might be one of an infinite set of
30 possibilities. The relationship could be, for example, that
31 the car listed in the second table is "owned" by the person
32 whose name is found next to a matching key in the first
33 table. On the other hand, it might be implied that the
34 matched person in the first table "drives" the car, or
35 "cleans" the car or has some other relation to the car. It
36 is left to the access control program to define what the
37 relationship is between entities in the first table and
38 entities in the second table.

1   It can be seen that relational database systems offer
2 users a great deal of flexibility since an infinite number
3 of relations may be defined (implied). Economy in
4 maintaining (updating) the database is also provided since a
5 change to a base table propagates through all other tables
6 which reference the base table. The access control program
7 of the database system can include information-updating
8 modules which, for example, change the key number in the
9 second table (Cars-Table) whenever ownership of a car
10 changes. If the name of the new owner is already in the
11 first table (Names-Table), it does not have to be typed a
12 second time into a new storage area and thus, extra work and
13 storage redundancy are avoided. The vehicle identification
14 number (VIN) remains unchanged. Minimal work is thus
15 expended on updating the database.
16   Despite these advantages, relational database systems
17 suffer from expandability and restructuring problems similar
18 to those of the above-described manual system. Sometimes
19 the rows within a particular table have to be altered to add
20 additional columns. This is not easily done. Suppose for
21 example, that a new government regulation came into being,
22 mandating that vehicles are to always be identified not only
23 by a vehicle identification number (VIN) but also by the
24 name and location of the factory where the vehicle was
25 assembled. If spare columns are not available in the
26 Cars-Table, the entire database may have to be restructured
27 to create extra room in the storage means (i.e. the disk
28 bank) for adding the newly required columns. New key
29 numbers will have to be entered into the new columns of each
30 row (e.g., a new "factory of assembly" key number) and
31 sorted in order to comply with the newly mandated regula-
32 tion. New search and inquiry routines will have to be
33 written for handling the newly structured tables.
34   In the past, much of this restructuring work was done
35 by reprogramming the computer at the object code or source
36 code level. This process relied heavily on an expert
37 programming staff. It was time consuming, costly and prone
38 to programming errors. Worst of all, it had to be redone

1 time and again as new informational requirements emerged
2 just after a last restructuring project was completed.
3 There is a need in the industry for a database management
4 system which provides quick responses to inquiries and which
5 can also be continuously updated or restructured without
6 reprogramming at the source or object code level.
7
8 SUMMARY OF THE INVENTION
9   It is an objective of the present invention to provide
10 a database system which is capable of storing voluminous
11 amounts of information, sifting through the information at
12 high speed, and is at the same time easily expandable or
13 restructurable to take on new forms of entities and
14 relationships.
15   In accordance with a first aspect of the invention, an
16 entity definition table (ENT.DEF) is defined within the
17 memory means of a computer system to store the name of an
18 allowed entity type (class) and the name of a single other
19 table (Entity-instances Table or "EiT" for short) where
20 instances of the allowed entity type may be stored. A
21 separate relationships definition table (REL.DEF) is defined
22 in the memory means to list in each row of the table:
23 (a) the name of an allowed relations type, (b) the name of a
24 single Relation-instances Table (RiT) where instances of the
25 allowed relationship type may be stored, (c) the name of a
26 primary (head) entity type to which the relation type may
27 apply and (d) the names of one or more secondary (tail)
28 entity types to which the named relationship may apply.
29 Each row of the Relation-instances Table (RiT) is provided
30 with at least one primary pointer which points to the
31 storage location of a first instance of the primary entity
32 type and at least one secondary pointer which points to the
33 storage location of a corresponding first instance of the
34 secondary entity type. Each row of the Relation-instances
35 Table (RiT) further includes a pointer to a relationship-
36 defining row in the REL.DEF table. The pointer can be the
37 name of an applicable relation type as recorded in the
38 REL.DEF table. Relationships between instances of a primary

1 entity and a secondary entity are thus expressly defined by

2 entries in the Relation-instances Table (RiT). Adding new

3 rows to this Relation-instances Table (RiT) allows for the

4 addition of new relations. Adding new rows to the REL.DEF

5 table allows for the creation of new classes (types) of

6 relationships. Since relation-defining tables can be

7 updated using a fixed set of update modules, reprogramming

8 at the source or assembly level is not needed for

9 restructuring the schema of the database.

10

11 BRIEF DESCRIPTION OF THE DRAWINGS

12   The invention will be described with reference to the

13 following figures in which:

14   Figure 1A is a block diagram of a conventional database

15 system.

16   Figure 1B is a timing diagram showing the delay between

17 the addressing and the delivery of storage data.

18   Figure 2A is a block diagram of a conventional key-

19 sequenced table organization.

20   Figure 2B is a block diagram of a conventional

21 relative-record table organization.

22   Figure 3 diagrams a multiple table system which is

23 based on a conventional relational database approach and

24 which has key-sequence organized tables.

25   Figure 4A is a conceptual diagram illustrating an

26 entity-relation schema in accordance with the invention.

27   Figure 4B is a further conceptual diagram of an

28 entity-relation schema according to the invention.

29   Figure 5 is a block diagram of an entity definition

30 (ENT.DEF) table in accordance with the invention.

31   Figures 6A and 6B are block diagrams of a relationship

32 definition (REL.DEF) table in accordance with the invention.

33   Figure 7 is a connection diagram showing how relations

34 may be explicitly defined in a Relation-instances Table

35 (RiT) so that unique relations between instances of a first

36 entity class and instances of a second entity class can be

37 identified.

38   Figure 8 is a block diagram of a database system

1 according to the invention.

2   Figure 9 is a block diagram of a relations processing

3 engine according to the invention.

4   Figure 10 graphs a variety of sample inquiry paths that

5 may be followed by the engine of Fig. 9.

6

7 DETAILED DESCRIPTION

8   The following includes a detailed description of the

9 best mode or modes presently contemplated by the inventor

10 for carrying out the invention. It is to be understood that

11 these modes are merely exemplary of the invention. The

12 detailed description is not intended to be taken in a

13 limiting sense.

14   Referring to Fig. 1A, the block diagram of a

15 conventional database system 100 is shown. The database

16 system 100 comprises a central processing unit (CPU) 110

17 which is operatively coupled so as to be controlled by an

18 access control program (object code) 120d stored in a first

19 memory means 120 (i.e., read-only-memory, ROM, or random

20 access memory, RAM). The CPU 110 in combination with the

21 first memory means 120 can be viewed as one or more machine

22 means for performing functions specified by the object code

23 120d. The CPU 110 is further operatively coupled to access

24 the data 130d of a "bulk storage" second memory means 130

25 also included in the database system 100. Individual

26 strings of digital information are represented by wiggled

27 lines (e.g., 120d, 130d) in Figure 1A. The bulk storage

28 means 130 typically takes the form of a large array of

29 magnetic disk drives, tape drives, or other mass storage

30 devices (e.g., arrays of Dynamic Random Access Memory [DRAM]

31 chips). The first (control) memory means 120 usually takes

32 the form of high speed RAM and/or ROM.

33   To access a particular string of data 130d stored

34 within the bulk storage means 130, the CPU 110 must provide

35 a corresponding address signal 131s (Figure 1B) in the form

36 of logic highs (H) and lows (L) to the bulk storage

37 means 130 over an address bus 131. As seen in the time

38 versus logic-level graph of Figure 1B, the address

1 signal 131s (usually an electrical signal) comprises a set

2 of logic high and logic low levels (H and L) transmitted in

3 a first time period t_0-t_1. There follows a second time

4 period, t_1-t_2, which is often referred to as an "access

5 delay", during which addressing circuits attempt to access

6 the addressed memory location. Depending on whether a

7 memory read or memory write operation is occurring, data

8 signals 132s are then transferred over a data bus 132

9 (Figure 1A) from the addressed location within the bulk

10 storage means 130 to the CPU 110 or vice versa during a

11 following third time period, t_2-t_3.

12   Referring still to Figure 1A, the object code 120d of

13 the access control program determines when and how the CPU

14 110 will access information 130d stored in the bulk storage

15 means 130. The CPU 110 issues address signals 121s (not

16 shown) over an address bus 121 to the first memory means

17 120, and in response, the first memory means 120 supplies

18 instruction signals 122s (not shown) over a data bus 122 to

19 the CPU 110. Information signals 122s can be exchanged

20 bidirectionally over data bus 122 between the CPU 110 and

21 the first memory means 120. Figure 1B may represent the

22 timing relation between address signals 121s and first

23 memory information signals 122s by replacing reference

24 numerals 131s and 132s with 121s and 122s, respectively.

25   It should be understood that neither the object code

26 120d of the first memory means 120 nor the data code 130d of

27 the mass storage means 130 is in human-readable form. A

28 translation machine is needed to convert the binary bit

29 strings of either memory means (120 or 130) into a form

30 which might be understandable to an experienced computer

31 programmer or to a lay computer user.

32   The object code 120d of the access control program is

33 produced by first generating (e.g., manually writing and

34 encoding) a source code listing 112 whose lines of

35 information 112d are usually understandable only to a highly

36 trained computer programmer. The source code listing 112

37 which is written in an assembly level or higher level

38 language (e.g., C, COBOL, FORTRAN, PASCAL, etc.) is

1 transformed into machine-readable form, and passed through a
2 first translation machine which may be referred to as a
3 compiler (or assembler) means 114. The compiler means 114
4 produces the machine-readable object code 120d according to
5 instructions provided by a machine readable version of the
6 source code listing 112. After it is stored in the first
7 memory means 120, the object code 120d is expressed as
8 machine detectable alternations (ones and zeroes) in a
9 physical attribute (e.g., voltage) of the medium which makes
10 up the first memory means 120. In this form, the object
11 code 120d is more readily convertible into data signals 122s
12 which are understandable to the CPU 110 than into
13 information which is understandable to a lay (non-
14 programmer) person. It is highly improbable that a lay
15 person will ever wish to access or understand or modify the
16 object code 120d stored within the first memory means 120.
17   The information strings 130d within the bulk storage
18 means 130 are similarly expressed as alternations in the
19 physical property of the storage medium making up the second
20 memory means 130. Some of the data strings 130d represent
21 "real" data which a lay-user may wish to access while others
22 of the strings 130d represent "ancillary" data such as
23 sequencing keys, threading pointers or control codes which a
24 lay-user is not interested in. The object code 120d of the
25 control program defines which is which.
26   When "real" data is to be extracted from the data
27 strings 130d within the bulk storage means 130, read and
28 understood by a lay person, a translation process similar to
29 compilation (or more correctly de-compilation) needs to take
30 place. Just like the compiler means 114 functions as a man-
31 to-machine translator, the combination of the first memory
32 means 120 and the CPU 110 defines a second man-to-machine
33 search-and-translate machine 115 which is used to search
34 through parts of the bulk stored data 130d, extract relevant
35 pieces of "real" data and convert the extracted data from
36 machine-readable form into human-readable form. The human-
37 readable output of the second translation machine 115 may be
38 produced in the form of a query output listing 150 (e.g., on

1 paper or on a video screen) as indicated in Figure 1A.

2   If a lay user (defined here as someone other than a

3 person who is an expert programmer familiar with details of

4 the source listing 112) wishes to obtain useful ("real")

5 information from the bulk storage means 130, the lay user

6 will normally supply a query input 140, in a form dictated

7 by a so-called "structured query language" (SQL) to the CPU

8 110. (In the illustrated example the user inputs the

9 request string "Please find all books having attribute xxx,"

10 where xxx could be the relations "author's last name

11 = Jones".) The combination of the CPU 110 and first memory

12 means 120 (which combination forms the second search-and-

13 translate machine 115) process this query input 140 and in

14 response, produces a series of address signals 131s which

15 are sent to the bulk storage means 130 and processes a

16 series of data retrievals 132s which eventually lead to the

17 production of a corresponding query output listing 150. (In

18 the example, it would be a listing of all books whose

19 author's name is "Jones".) The access control program 120d

20 is charged with the task of enabling various types of

21 queries 140 and making sure that the queries do not violate

22 basic rules of logic.

23   When the information 130d within the bulk storage means

24 130 needs to be updated, by for example adding new books, a

25 similar exchange occurs between the translating machine 115

26 and a lay user. The lay user supplies an update input 160,

27 again as dictated by a pre-specified structured query

28 language (SQL), and in response, the translating machine 115

29 rearranges the data 130d within the bulk storage means 130

30 to achieve the requested update.

31   Referring to Figure 2A, a first embodiment 200 of the

32 data base system 100 will be described in more detail.

33 Figure 2A schematically illustrates a section 130a of the

34 bulk storage means 130 according to embodiment 200 wherein

35 some of the stored data strings 130d are arranged to define

36 a key-sequenced type of table. In a first record region

37 (Record No. 1) of the table 130a there is provided a first

38 continuous data string 230 which is subdivided to have a

1 first string portion 231 representing an author's name

2 (illustrated as the contents of a rectangular box), a second

3 string portion 232 contiguous thereto for representing a

4 name threading pointer (illustrated as a second rectangular

5 box coupled to the first rectangular box by an address

6 proximity link P_{11}), a third data string portion 233

7 representing the book's title (which is linked to the second

8 portion 232 by proximity link P_{12}), a fourth subsection 234

9 representing a title threading pointer (linked to box 233 by

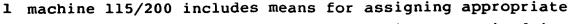10 address proximity link P_{13}), a fifth subsection 235 representing

11 the book's location (linked to box 234 by proximity link P_{14}) and

12 a sixth subsection 236 representing a location threading

13 pointer (linked to box 235 by proximity link P_{15}).

14   The name threading pointer 232 is located directly

15 adjacent to the author's name subsection 231 within the

16 address space of Record No. 1, as indicated by address

17 proximity link P_{11} and thus, there is an "implied" logical

18 connection between the data contents of boxes 231 and 232.

19 The book's title subsection 233 is located directly adjacent

20 to the name threading pointer 232 as indicated by address

21 proximity link P_{12}. The combined, proximity linkage, P_{11}-

22 P_{12}, "implies" a relationship between the contents of boxes

23 231 and 233, namely that they apply to various attributes of

24 a common book. This format repeats for data subportions

25 234-236. Only boxes 231, 233 and 235 contain "real" data

26 which is useful to a lay person. The other boxes, 232, 234

27 and 236 of Record No. 1 contain "ancillary" data which is

28 useful to the search machine 115 but does not provide the

29 kind of "real" information sought by an inquiring lay

30 person.

31   The implied relations between the "real" data boxes,

32 231, 233 and 235 of Record No. 1, arise only after "meaning"

33 is assigned to all the boxes 231-236. Such "meaning" comes

34 from the operation of the search-and-translation machine 115

35 (Fig. 1). To understand this concept, assume that an

36 automated "searching" machine (computer) 115/200 of

37 embodiment 200 is examining the data string 230 held within

38 the single Record No. 1. Assume further that this searching

1 machine 115/200 includes means for assigning appropriate
2 "meanings" to each of the data subportions contained in each
3 of subsections 231-236 to thereby designate some as
4 containing "real" data and others as containing "ancillary"
5 (e.g., pointer) data. In that case the search machine
6 115/200 can scan horizontally across the record, parse the
7 data string 230 into subsections of appropriate size and
8 extract the name of the book's author, the book's title and
9 the location of the book within the library, as desired. On
10 the other hand, if the searching machine 115/200 does not
11 possess information which tells it that box 232 is a
12 threading pointer, box 233 is a title, etc., then all boxes
13 will look alike to the search machine, there will be no
14 "meaning" assigned and the search machine 115/200 will not
15 be able to extract a desired piece of data. Thus, while not
16 shown in Fig. 2A, it is to be understood that there is a
17 cooperative relation between how the object code 120d of the
18 search machine 115/200 causes that search machine to access
19 the parts of bit string 230 via the signal busses, 131 and
20 132, how subportions of bit string 230 become designated as
21 "real" or "ancillary" data, and how relations are implied
22 between separate pieces of real data. The structure,
23 meanings inter-relations between the parts of bit string 230
24 are intimately linked to the structuring of the object code
25 120d.
26   In Fig. 2A, the bulk memory means section 130a is shown
27 to include additional record areas (Record No. 2, Record
28 No. 3, etc.) each having the same data structure (repre-
29 sented respectively as string 240 which comprises data sub-
30 sections 241-246 and string 250 which comprises data
31 subsections 251-256). Although Record No. 1 is in physical
32 proximity with Record No. 2, as indicated by physical (or
33 address) proximity link PR_{12}, and Record No. 2 is in
34 physical proximity with Record No. 3 as indicated by
35 physical proximity link PR_{23}, the data items (231-236, 241-
36 246, 251-256) within each record do not need to be examined
37 according to this physical ordering. Instead, the name
38 threading pointer 232 of Record No. 1 can represent the

1   address of any other arbitrary record area within the bulk

2   storage means section 130a whose author's-name will serially

3   follow the author's-name of box 231 during a search

4   process. This is represented in Figure 2A by the dashed

5   logical link L_{11} which points to some arbitrary record area,

6   Record.Addr.$_{11}$ of section 130a. The name threading pointer

7   of the referenced record, Record.Addr.$_{11}$, can point to yet

8   another arbitrary record. With this mechanism, a list which

9   is sorted (alphabetically for example) according to author's

10 last name may be formed even though the records are not

11 physically ordered in any specific sequence. The list is

12 referred to as a "key-sequenced" list in cases where, as

13 here, the sequencing key (or sort key) is data stored in the

14 boxes e.g., 231, 241, 251, etc., of a table column.

15   The title threading pointers (234, 244, 254) of each

16 record may be used to form a different key-sequenced path in

17 which books are examined according to subject matter or

18 alphabetically according to the book's title or according to

19 some other ordering algorithm. The location threading

20 pointers (236, 246, 256) can be similarly used to create a

21 key-sequenced list which will identify what book is

22 physically located next to what other book on the library's

23 shelves.

24   For the sake of illustrative simplicity, only one

25 threading pointer (i.e., 232) is shown attached to each real

26 data item (i.e. 231) of each record, but it should be

27 apparent that the author's name 231 may have many threading

28 pointers, one for threading alphabetically according to last

29 name, and others for threading according to additional

30 relations such as geographic location, age, number of

31 published books and so forth. It is up to the computer

32 programmer and the access control program 120d to assign

33 "meaning" to each box and thus determine whether that box

34 will function as a storage area for real data or for

35 ancillary data such as pointer data.

36   The records of Figure 2A may be visualized as being

37 serially stacked one on the next according to a sequence

38 defined by a preselected one of the threading pointers (e.g.

1 232 or 234 or 236) to thereby create a displayable table
2 which has as entries in the columns of each row, the real
3 data items: author's name 231, book's title 233 and book's
4 location 235. The ancillary threading pointers 232, 234,
5 236 are hidden from the lay user's view. New rows are added
6 to the table by breaking a logical link (e.g., L_{11}) between
7 a preceding pointer (e.g. 232) and a next pointer (e.g. 252)
8 to insert a new record in the search path. The rows can be
9 of variable length since the linking address pointers can
10 point to any arbitrary location in the bulk memory means
11 130. To get to the Nth item of a threaded list, one
12 normally sequences from the beginning of the list (table)
13 through all the threading pointers until the Nth access is
14 performed, at which point the contents of the addressed
15 record area can then be read. For relatively large tables
16 (e.g. those having thousands of rows), this process of
17 sequencing through all the threading pointers to reach the
18 N^{th} row of a table can take a significant amount of time.
19   Referring to a second embodiment 260 shown in
20 Figure 2B, the structure of an older and less sophisticated
21 data organizing system will be described. In a bulk memory
22 section 130b of this older system 260, data is organized
23 according to what is commonly referred to as "relative
24 table" addressing. Threading pointers are not used for
25 logically linking one record (row) to the next. Instead,
26 each data string (e.g., 270) can be shrunk to contain only
27 the essential target information, such as in this example,
28 author's name (271), book's title (273) and book's location
29 (275), with one item of real data being physically located
30 adjacent to the next. The examination of all record items
31 in this structure 260 may be performed according to the
32 physical location of each record (270) within the address
33 space of bulk storage area 130b (the next adjacent string
34 280 follows first string 270 and so forth). Unlike the
35 purely key-sequenced organization of Fig. 2A, the physical
36 proximity links PR_{012}, PR_{023}, PR_{034}, etc., of Fig. 2B do
37 indicate a particular ordering of the stored information.
38   The relative-table organization is somewhat similar to

1 the way that index cards are physically ordered in a manual
2 library system according to author's last name, except that
3 the library catalog trays should now be visualized as having
4 sequentially arranged grooves defined on their bottom-inner
5 surfaces. Each groove is numbered according to its absolute
6 position and only one card can be slotted into each
7 groove. With this system, each card can be immediately
8 located by its groove number rather than by thumbing through
9 the information of all previous cards. If a groove number
10 is known, substantial time can be saved in locating the
11 corresponding card and obtaining the information written on
12 its face. If the groove number is not known, the same
13 relative-table organization can be searched by sequentially
14 thumbing through the trays and examining the cards according
15 to a key-sequenced approach in order to find a desired card
16 even though the cards are stored in grooves. The relative-
17 table organizing method is not mutually exclusive of a
18 key-sequenced examination method. There is a difference
19 between a purely key-sequenced table and a relative table,
20 however. A relative-table organized system is not as easily
21 updated as is a purely key-sequenced system. In the
22 relative table system, a new card cannot be inserted between
23 two cards which already fill adjacent slots. This
24 inflexibility has led many in the database management field
25 away from the relative-table method and towards purely
26 key-sequenced systems since the latter can accept any number
27 of new cards for insertion between old cards.
28   In Fig. 2B, all the record areas are of a fixed and
29 predefined length. The fixed length of each record defines
30 the groove size. To access the Nth item of a "relative-
31 table" type of list 130b, one need only multiply the fixed
32 record length by the value N to directly obtain the physical
33 address (slot) of the desired record. There is no need to
34 sequence through a chain of threading pointers in order to
35 find a desired row once its slot number (groove number) is
36 known. Empty slots 290, such as the slot number 4 shown in
37 Figure 2B, are preferably scattered throughout the address
38 space of the bulk memory section 130b to allow for

1 occasional insertion of new items.

2   It should be noted that while the relative table

3 organization 130b of Figure 2B is neither as flexible nor as

4 easily updated as the key-sequenced organization 130a of

5 Figure 2A, the relative-table structure 130b has one major

6 advantage over the key-sequenced structure 130a; an Nth item

7 in a relative-table list 130b may be accessed much faster

8 than the Nth item of a key-sequenced list 130a.

9   Figure 3 is a block diagram of a bulk storage area 130c

10 whose data 130d is organized according to a known key-

11 sequenced scheme which is often referred to in the industry

12 as a "relational" database. A "tables" area 300 contains a

13 plurality of tables 310, 320, 330, 340 and 350. Each of

14 these tables is defined purely by a threaded-list, key-

15 sequenced structure such as shown in Fig. 2A. For the sake

16 of illustrative brevity the list threading pointers (i.e.,

17 232, 234, 236) are not shown. Only the non-threading boxes

18 (i.e., 231, 233, 235) are shown.

19   Rows are illustrated to extend horizontally (in the "x"

20 direction) in Fig. 3 while table columns are illustrated to

21 extend vertically (in the "y" direction). Each table

22 310-350 is shown to have its respective rows sorted

23 numerically according to "key" numbers that are stored in

24 its leftmost column (referred to here as the "sort column").

25   A first of the key-sequenced tables, 310 (also labeled

26 "Table of Names"), is shown to have two columns. One (right

27 side) column 312 holds "real" data representing the names of

28 various persons while a preceeding (left side) column 311

29 holds unique key-numbers, 1, 2, 3, ..., N, N+1, N+2, ...,

30 each associated with a unique name of a person. The

31 association of a person's name to a key-number is "implied"

32 by the fact that the key number 1, 2, 3, ..., N, ..., is

33 located in the same row of table 310 as is the corresponding

34 "Person's Name". Each key-number of left column 311 is

35 referred to as a "Name Identification Number" (abbreviated

36 here as N-IDN). Table 310 is shown to have been pre-sorted

37 according to the N-IDN's of column 311. The sorting method

38 is indicated in Fig. 3 by positioning the initials "KSO"

1 over column 311 to tag that column as the Key-Sequenced-
2 Ordering column of table 310.
3   To find the name of a person within table 310 whose
4 associated identification number is known to be N, one
5 normally starts at row number 1 of the left column 311,
6 where the N-IDN of the first person's name is stored and
7 threads downwardly (in the y direction) through the
8 threaded-list pointers (not shown) associated with this sort
9 column 311, testing each corresponding entry of column 311
10 for a match until the position holding the number N is
11 found. Then one moves horizontally (in the x direction)
12 across that row to the right column 312 to extract the name
13 associated with the Nth name identification number (N-IDN).
14   When an automated search machine 115 performs this
15 thread and test process, it must retrieve data from the
16 memory area 130c at least N times before the target data
17 (Person's-Name) is retrieved. The time for retrieving the
18 target data is thus at least N times the access delay period
19 (e.g., the $t_2 - t_1$ period of Fig. 1B) of the memory means
20 130. By way of example, if N = 1000 and the access time of
21 memory means 130 is 30 milliseconds, then it can take 30
22 seconds or more just to retrieve one name. If a thousand
23 names are to be randomly retrieved at different times from
24 the range N, N+1, N+2, ..., N+M (where M would be 1000 or
25 higher), then it can take as much as 30,000 seconds (8.3
26 hours) or longer just to perform this simple table look-up
27 task.
28   The N-IDN field of each row is generally made much
29 shorter in bit length than its associated Person's-Name
30 field. The N-IDN can be viewed therefore as an abbreviation
31 of a person's full name. The first table 310 can be viewed
32 as a conversion list or look-up table which allows one to
33 easily convert a given abbreviation (N-IDN) into a full
34 name.
35   A second, separate, table 320 (also labeled as "Table
36 of Locations") is shown to contain two similar columns.
37 Right column 322 stores "Home Addresses" in full while left
38 column 321 holds unique, Home-Identification-Numbers

1 (abbreviated H-IDN) which are generally shorter in bit
2 length than the associated "Home-Address" fields. The
3 H-IDN'S thus can serve as abbreviations for the full address
4 fields. Table 320 is ordered numerically according to the
5 H-IDN's as indicated by the legend "KSO" over column 321.
6 The table 320 can therefore easily serve as part of an H-IDN
7 abbreviation to full address converting means.
8   Since many people often live at a single home address,
9 it is plausible that a single home address will be shared by
10 persons of different names. Relational database theory
11 recognizes this and teaches to separate information (e.g.,
12 home address) that might be shared by many entities away
13 from any unique one of those entities (e.g., person's
14 name). Table 310 is accordingly separated from table 320.
15 Concurrently, it should be possible to relate a person's
16 full name to a full home address without having to
17 repeatedly duplicate the full name string or full address
18 string within the bulk storage means 130. The data
19 organization 300 shown in Figure 3 includes a third key-
20 sequenced table 330 which is structured for doing just that;
21 linking one persons' name with one home address while using
22 the abbreviated bit strings, N-IDN and H-IDN.
23   Third table 330 comprises three vertical columns, 331,
24 332 and 333. Left column 331 holds Person Identification
25 Numbers (P-IDN's), 1, 2, 3, ... , P. The rows of third
26 table 330 are sorted using the P-IDN's as the sort key. For
27 each row of the third table 330, the second column 332
28 contains a Name-IDN and the third column 333 contains a
29 Home-IDN. Each Name-IDN of third table 330 (for example, at
30 row 4 of table 330 whose column 332 contains the value "N")
31 should have in the left column 311 of the Names table 310 a
32 matching key number (e.g., the number N which is pointed to
33 by arrow L_{41}). Thus an N-IDN stored in the third table 330
34 can be used to indicate the row within the first table 310
35 where a person's full name may be found. Each Home-IDN of
36 the third table 330 should similarly have a matching key
37 number (e.g., the number 2 which is pointed to by arrow L_{43})
38 within left column 321 of the second "Locations" table 320

1   between the Person's-Name field stored at row N of table 310

2   and the Home-Address field stored in row 2 of table 320.

3   Arrow L_{zz} denotes that all illustrated linkages (L_{41}-L_{48}) in

4   Fig. 3 spring forth from randomly-distributed object code

5   modules 120d* of the access control program 120d. Note that

6   the third table 330 assumes by its three column structure a

7   one-to-one cardinality between person-name and home-

8   address. It is assumed that a person can have only one home

9   address. The structure of table 330 is incapable of

10 handling a situation where a person has, for example, both a

11 summer home-address and a winter home-address. Restructur-

12 ing of the third table 330 would be called for if it becomes

13 desirable to associate each person's name with more than one

14 home address.

15   A number of advantages come from organizing the tables

16 of data storing area 300 separately according to relational

17 database theory. Storage space is conserved in cases where

18 plural entities of a first type (person) are related to a

19 common entity of a second type (home address). The same

20 Home-IDN can appear many times down column 333 without

21 consuming large amounts of memory space while the actual

22 full address is stored only once in second table 320. When

23 a person moves to a new home address, the corresponding

24 Home-IDN in column 333 can be easily altered to point to a

25 new position within the second table 320 which contains the

26 new home address (e.g., H+1) thereby implying the new

27 person-to-address relation. If a person changes their name

28 (i.e., by way of marriage) the home address can remain the

29 same. Only the first table 310 needs to be modified and

30 updating work is thus minimized. Also, the basic listings

31 "Names" 310 and "Addresses" 320 can be used to imply a wide

32 variety of "relations" other than a relation between a

33 person's name and his/her home address using the same

34 abbreviated set of identification numbers (IDN's).

35   By way of example, assume that the first three tables,

36 310, 320 and 330, are used by a business institution

37 (company) to keep track of the names of their employees and

38 the corresponding home addresses of these employees. Let it

1   be supposed· that many employees need to commute to work by a
2   privately-owned car. Some employees drive their own car,
3   some drive a car owned by another employee and some are
4   merely passengers. Let it be further assumed that after
5   tables 310, 320, 330 are defined in a mass storage means
6   130, the company decides to also keep track of which person
7   drives which car, which person is a passenger in which car
8   and further, who the owner of the car is.
9   A fourth table 340 (Table of Drivers) may be
10   constructed as shown in Fig. 3 to have a first key-sequenced
11   column 341 storing plural driver identification numbers
12   (abbreviated here as D-IDN's), 1, 2, 3, ..., D. A second
13   column 342 is provided for holding person identification
14   numbers (P-IDN's) and a third column 343 is provided for
15   holding car identification numbers (C-IDN's). A fifth table
16   350 (Table of Cars) may be similarly constructed as shown
17   with a first KSO column 351 for holding the C-IDN's (1, 2,
18   3, ..., C), with a second column 352 for holding owner
19   identification numbers (O-IDN's) which will point to the one
20   person who owns the vehicle and with a third column 353 for
21   holding a vehicle serial number (SN). While not shown, it
22   should be apparent that a sixth table (Table of Passengers)
23   would be constructed with the same organization as that of
24   fourth table 340 to identify passengers and their
25   corresponding car.
26   Referring to row D of table 340, it can be seen that
27   one implied link L_{44} identifies driver D as being the person
28   of P-IDN=4 who has the name implied by earlier link L_{41} and
29   the home address implied by earlier link L_{43}. Proximity
30   link P_{45} implies that driver D drives the car having
31   C-IDN=2. The latter number implies a logical link L_{46} to
32   row 2 of table 350 which holds the serial number (SN) of the
33   driven car. By way of another proximity link, P_{47}, in row 2
34   of the same fifth table 350, a further logical link, L_{48},
35   indicates that the owner of car C-IDN=2 is the person
36   P-IDN=P of table 320. It was assumed by the structure of
37   table 350 that each car can have only one owner and one
38   serial number.

1   Consider, however, what happens if a new government
2 regulation comes into being allowing for more than one owner
3 per car or requiring multiple identification numbers for
4 each car. The fifth table 350 may have to be restructured
5 to add new columns (i.e., 354, 355, etc.; not shown) which
6 would allow for the implication of such new relations. This
7 means that the access control modules 120d* which define the
8 "meaning" of each data field (subsection) within table 350
9 would have to be revised. Referring back to Fig. 1 it can
10 be seen that modification to the control code 120d* will
11 usually occur first in the original source code 112, which
12 is then compiled 114 as indicated in Fig. 1, debugged to
13 correct programming errors (not shown) and thereafter
14 repeatedly compiled 114 and debugged until all apparent
15 errors are removed. The process of restructuring relations
16 within a relational-type database system (300) therefore
17 tends to be time-consuming, costly, and prone to error.
18   A newer form of database organization, referred to
19 sometimes as the "object oriented" approach, has been
20 proposed to solve some of the problems associated with
21 reorganizing and updating previous database systems.
22 According to the object-oriented approach, encapsulation
23 bubbles are defined to hide from external view, data which
24 is encapsulated within the bubble. Each bubble is referred
25 to as an "object" and the encapsulated information of the
26 object is referred to as the object's "attributes." One
27 bubble may encapsulate a second bubble which in turn
28 encapsulates third, fourth and further bubbles so that a
29 relatively complex data structure may be defined. Objects
30 can be assigned to "classes" and by such assignment they can
31 be made to automatically "inherit" the attributes of other
32 objects in the same class, even when the class attributes
33 are changed after creation of the objects.
34   There is still controversy in the field over what
35 constitutes "object oriented" and how such a concept may be
36 practically applied to database management systems.
37 Experimental versions of object-oriented systems are often
38 too slow in performing update and inquiry servicing to be

1 practical in commercial settings. The present invention
2 takes an approach which might be considered a partial hybrid
3 of the object-oriented approach and the earlier-described
4 relational database methodology. It provides a database
5 system which is capable of operating at commercially
6 acceptable speeds and which is easily restructured as well
7 as updated. The invention will be explained first
8 conceptually and then by concrete examples.
9   Referring to Figure 4A, there is shown a relational
10 graph or "schema" 400 which contains three egg-shaped
11 bubbles labeled respectively as "Customer", "Address" and
12 "Account". These bubbles are not intended to represent
13 "objects" from the object-oriented school of thought, but
14 rather "classes" of entities. Each of these bubbles is
15 referred to as an "entity type" or "entity class". The
16 "Customer" entity class generically covers all entities
17 which might fit under the broad descriptor "Customer",
18 regardless of whether that entity is a natural person, a
19 business corporation, an association or so forth. The
20 "Address" entity class covers all entities which fit under
21 the broad descriptor "Address" regardless of whether the
22 subject entity is a residential address, a business address,
23 a post-office mailing address or so forth. Similarly, the
24 "Account" entity class covers all sorts of accounts
25 including savings accounts, checking accounts, trust
26 accounts, etc.
27   Each entity bubble may contain one or more "instances"
28 of the entity class (i.e., Customer, Address, Account) which
29 it represents. By way of example, let it be assumed that
30 there are three customers whose names are "Customer-A",
31 "Customer-B" and "Customer-C". Let it be further supposed
32 that because of a peculiar rule, the Customer bubble (also
33 labeled as entity class "E-1") is restricted to contain the
34 name of only one customer at a time, say "Customer-B", while
35 the address bubble (E-2) can at the same time contain many
36 "addresses", each corresponding to that Customer-B. If
37 Customer-B is a person, the address instances might be
38 summer-home and winter-home addresses. If Customer-B is the

1   name of a business having a chain of stores, the plural

2   addresses in the second bubble (E-2) might be the mailing

3   addresses of those stores. The name "Customer-B" is an

4   example of a first instance, $I_{1/E1}$, of the E-1 entity class

5   and is illustrated conceptually in Fig. 4A as a small sphere

6   $I_{1/E1}$ enclosed in the entity class bubble E-1. Three

7   instances, $I_{1/E2}$, $I_{2/E2}$ and $I_{3/E2}$ of entity class E-2 are

8   similarly illustrated as three spheres inside of entity

9   bubble E-2. It is also assumed here that the Account bubble

10 (E-3) is restricted by a peculiar rule so that at any one

11 time it may contain only one account number (instance $I_{1/E3}$)

12 which is somehow associated with Customer-B.

13   Until now we have been visualizing the instances,

14 $I_{1/E1}$, $I_{1/E2}$, $I_{2/E2}$, $I_{3/E2}$ and $I_{1/E3}$ of respective entity

15 classes, E-1, E-2 and E-3 as isolated spheres floating

16 separate from one another, without identifying any specific

17 relation between the instances. The present invention

18 treats "relations" as being objects of equal substance to

19 the entities they tie together. There are relation

20 "classes" and instances of a specified relation class.

21 Three arrow-shaped bubbles, R-1, R-2 and R-3, are shown in

22 Figure 4A to be respectively coupling the Customer entity

23 class (E-1) to the Address entity class (E-2), the Account

24 entity class (E-3) to the Customer entity class (E-1) and

25 the Account entity class (E-3) to the Address entity class

26 (E-2). These linking bubbles (R-1, R-2, R-3) are referred

27 to here as "relationship" types or classes. Each relation

28 bubble R-x (where x is an arbitrary identifier, 1, 2, 3,

29 etc.) is visualized as having a bulb-shaped Head portion, H,

30 an elongated body portion B and an arrow-shaped Tail

31 portion, T. A "Head attribute" can be assigned by each

32 relation bubble R-x to the entity bubble (E-h) located at

33 its head end (H). A "Tail attribute" can be correspondingly

34 assigned by each relationship bubble R-x to the entity

35 bubble (E-t) located near its tail end (T). The combination

36 of the Head-attribute, if any, plus the Tail-attribute, if

37 any, can be used to give the relationship bubble (R-x) a

38 "meaning". This meaning is generated by associating with

1 the body portion B of each relationship bubble (R-x), a

2 "meaning-string" which preferably, but not necessarily, has

3 a head character-string and a tail character-string. The

4 combination of an "entity-class name" (ECN-h) associated

5 with the head entity type (E-h), the meaning-string (M-s) of

6 the connecting relation type (R-x) and another entity class

7 name (ECN-t) associated with the tail entity type (E-t) are

8 concatenated according to the formula, (ECN-h)+(M-s)+

9 (ECN-t), to expressly define a relational phrase. The

10 expressly defined phrase can be modified by changing any one

11 or all of its three components; (ECN-h), (M-s) and (ECN-t).

12   In more concrete terms, the top relation bubble R-1 is

13 shown to have the meaning string " 's business". The

14 substring, "'s" is a head character-string while the

15 substring "business" is a tail character string. By itself,

16 the meaning-string ('s business) appears to be nonsensical,

17 but in conjunction with the class names its head and tail

18 entities, E-1 ("Customer") and E-2 ("Address"), this first

19 relations bubble, R-1, forms the relational phrase: "The

20 Customer's business Address". Instance $I_{1/E1}$ is a specific

21 customer's name (i.e., "Customer-B") and instances $I_{1/E2}$,

22 $I_{2/E2}$ and $I_{3/E2}$ are now defined as specific instances of

23 that customer's business addresses (i.e., the addresses of

24 individual stores in a chain of stores owned by Customer-B).

25   Of importance, it is to be noted that the first entity

26 bubble, E-1 (Customer), does not itself encapsulate the

27 attribute of possession as indicated by the apostrophed head

28 character-string "'s". Instead, that attribute of

29 possession is encapsulated by the first relationship bubble,

30 R-1. Furthermore, the second entity, E-2 (Address), does

31 not encapsulate the modifying attribute "business". Instead

32 that attribute is also encapsulated by the relation bubble

33 R-1. Thus, each entity bubble (E-1, E-2, E-3) is free of

34 any narrowing attributes or modifiers and instead,

35 represents a relatively broad and generic listing of data

36 items which can come under the heading of either "Customer"

37 or "Address" or "Account". The advantage of this structure

38 will become apparent shortly.

1   Consider for a moment what happens if the meaning-
2 string in relation bubble R-1 is changed from " 's business"
3 to " 's headquarters". Under this circumstance, the rules
4 change. The address bubble (E-2) should be restricted to at
5 any one time contain only a single instance (e.g., $I_{1/E2}$)
6 representing the "Customer's headquarters Address" rather
7 than many instances. Presumably each customer can have only
8 one headquarters address. Thus, the "cardinality" of
9 relations bubble R-1 must be changed from its earlier one-
10 to-many {1:m} format, as was possible with business
11 addresses, to a one-to-one setting {1:1}. According to the
12 invention, each relation bubble, R-x, has a cardinality rule
13 (e.g., {1:1} or {1:m}) associated with its body B as well as
14 a meaning- string (e.g., "'s business").

15   Consider, next what happens in a business database if
16 users are allowed to enter a customer name but leave out the
17 mailing address or telephone number of that customer. Most
18 companies operate under a strict rule which requires its
19 office workers to record at least one forwarding address or
20 telephone number when the name of a new customer is
21 entered. To enforce this requirement, each relation bubble
22 (R-1) further incorporates a mandatory-coupling character
23 which can be either "Y" or "N" (representing yes or no). If
24 it is required that at least one instance ($I_{1/E2}$) of a tail
25 entity class E-2 should be created whenever an instance
26 ($I_{1/E1}$) of a head entity class E-1 is created, then the
27 mandatory-coupling character of relation bubble R-1 is set
28 to "Y". This indicates that instance $I_{1/E1}$ should not exist
29 without instance $I_{1/E2}$. The "MC" lightning bolt shown
30 emanating from $I_{1/E1}$ represents this mandatory coupling of
31 instances. On the other hand, if such coupling is not
32 mandatory, the coupling character is set to "N" and there is
33 no "MC" connection.

34   As further examples of the concepts behind the
35 invention, the second relation bubble, R-2, is shown to
36 contain in Fig. 4A the meaning string, " 's owning", the
37 cardinality rule, {1:1}, and the mandatory-coupling
38 character, "Y" (presumably every account should have an

1 owner). The third relation bubble, R-3, is shown to contain
2 the meaning string, " 's statement mailing", the cardinality
3 rule, {1:1}, and the mandatory-coupling character, "N"
4 (presumably an account holder can pick up his/her statement
5 rather than having it mailed). Instances of entity E-1
6 which satisfy the relationship created by relation bubble
7 R-2 are read as "The Account's owning Customer". Instances
8 of entity E-2 which comply with the relationship created by
9 relation bubble R-3 satisfy the descriptive phrase,
10 "Account's statement mailing Address", or stated otherwise,
11 the address to which account statements are mailed for the
12 particular instance $I_{1/E3}$ of the Account entity class E-3.
13   By changing the meaning-string within a relation bubble
14 R-x, it is possible to create new relational phrases
15 although the Head and Tail entity classes remain the same.
16 By changing either or both of the Head and Tail entity
17 classes (E-h or E-t), it is possible to again create new
18 relational phrases although the relation bubble R-x remains
19 unchanged.
20   Consider what happens for example when the meaning-
21 string of relation bubble R-3 is changed to the phrase:
22 "which is managed at bank branch having". Then the
23 combination of the class names or meanings associated with
24 entity bubble E-3, relation bubble R-3 and entity bubble E-2
25 provides for an inquiry path allowing one to find the
26 Account which has a specific bank branch address as its
27 managing branch. Consider what happens if the tail portion
28 T of relation bubble R-3 where moved from E-2 to a new
29 entity bubble (not shown) which is labeled "Managing
30 Officer" rather than "Address". Then the relational phrase
31 becomes "Account which is managed at bank branch having
32 [this person as its] Managing Officer". It can be seen that
33 an entirely different inquiry path is formed with each
34 change of a head entity type, tail entity type or relation
35 type.
36   Inquiry paths can be defined to extend through
37 pluralities of entity and relation bubbles as well as
38 between just two entity bubbles. Still referring to Fig.

1 4A, suppose that a bank officer finds an important document
2 bearing only an account number on it. The bank officer
3 needs to immediately contact a person who is authorized to
4 manage that account for more details about the document. In
5 such a case, the bank officer would turn to a database
6 processing engine according to the invention (explained
7 later with reference to Fig. 9), start at the known instance
8 of the account number, $I_{1/E3}$, which is shown contained
9 within the Account bubble (E-3), jump through the relation
10 bubble R-2 ('s owner) to the Customer bubble (E-1) in order
11 to learn who the owning customer is (instance $I_{1/E1}$) and
12 then with that new information ($I_{1/E1}$) serving as a stepping
13 stone, jump from the Customer bubble (E-1) through the
14 relation bubble R-1 ('s business) to the Address bubble
15 (E-2) to learn the address at which he may contact the
16 account manager. This is merely an example, inquiry paths
17 can include many more bubbles, they can branch out to form a
18 tree rather than being serial and they can produce many
19 pieces of information which are useful for solving a puzzle
20 rather than just one piece of target information.
21   Relation bubbles (R-x) do not have to be single
22 tailed. Referring to Fig. 4B, further variations of the
23 concept behind the invention are illustrated. A fourth
24 relation bubble, R-4, is shown to have a plurality of tail
25 ends, T1, T2 and T3, so that a single meaning-string (e.g.,
26 "'s business") can simultaneously couple a common Head
27 entity (Customer) to a plurality of Tail entities (e.g.,
28 Address, Account and Telephone). Moreover, a relation
29 bubble does not need to span between different entity
30 bubbles. Figure 4B shows another relation bubble, R-5,
31 which folds back in a loop so that the Head entity
32 (Customer) is also the Tail entity. In the illustrated
33 example, the relation bubble R-5 contains the meaning string
34 "'s largest". Given the name of a first customer, this
35 back-looping relation bubble R-5 allows one to find that
36 customer's largest customer. The loop may be followed
37 around ad infinitum to obtain a long list of largest
38 customers belonging to other largest customers.

1   With the above-mentioned conceptual models in mind, a

2 concrete embodiment of the invention now will be constructed

3 piece by piece. Referring to Figure 5, there is shown a

4 first table 500 which is referred to as an entity definition

5 table or in abbreviated form, ENT.DEF Table 500. This

6 entity definition table 500 is stored within a data storing

7 area 130-RP of a database engine in accordance with the

8 invention. Data storing area 130-RP preferably resides

9 within a bulk storage means 130* such as diagrammed in

10 later-to-be described Fig. 8. Unlike the earlier described

11 tables 310-350 of the relational system shown in Fig. 3,

12 which relied on a purely key-sequenced organization, the

13 entity definition table 500 of Fig. 5 can rely on a relative

14 table organization (abbreviated here as "RTO") which

15 features faster data access properties and is also adaptable

16 to key-sequenced search algorithms (but not key-sequenced

17 update methods). Each row of the ENT.DEF table 500 is of a

18 fixed bit length and has two columns. The first (left)

19 column 500a stores a two character field (e.g., "CU," "AD,"

20 "AC" or "SU") which is an abbreviation of an entity class

21 name. The abbreviation "EA" will be used here to mean "the

22 abbreviated form of the entity class name" (Entity-name

23 Abbreviation). By way of example, slot number 1 is shown to

24 contain the two-character abbreviation "CU" (representing

25 the entity name "Customer") in its left column 500a.

26   For expedience sake, a matrix notation is used here to

27 identify the columns of table 500 with letters, a, b, c,

28 ..., etc. and the rows with a numeral preceeded by a

29 period. The symbol 500a.1 thus refers to the box in table

30 500 at column 500a and row 500.1.

31   As further seen in Fig. 5, the abbreviation "AD" is

32 stored in box 500a.2 to represent the entity name

33 "Address". Box 500a.3 holds the abbreviation "AC" for

34 "Account" and box 500a.4 stores the abbreviation "SU" for

35 "Supplier". The slot or row numbers, .1, .2, .3 and .4 of

36 table 500 do not occupy storage space within memory means

37 130-RP. They merely represent the physical or logical

38 address of their respective rows, 500.1, 500.2, 500.3 and

1 500.4.

2   In the corresponding right column 500b of the ENT.DEF
3 table 500 there is stored, for each slot (.1, .2, .3, .4,
4 etc.) the name of a single other table where instances of
5 the named entity class are stored. The abbreviation "EiT"
6 (Entity-instances-Table) will be used here to mean the table
7 where instances of the entity class are stored. Again by
8 way of example, box 500b.1 is shown to reference an EiT
9 called "T.Companies" as the single table where instances of
10 the entity class "Customer" are stored. The entry in box
11 500b.2 is "T.Addresses" and the entry in box 500b.3 is
12 "T.Accounts". Note that the entry in box 500b.4 is
13 "T.Companies" just as it is for box 500b.1. Instances
14 belonging to two different entity classes (e.g., "CU" and
15 "SU") may be stored in one instances table (EiT) under
16 situations where the data structures of the instances are
17 compatible to the structure of that EiT (e.g., the entity
18 instances table has enough columns of appropriate widths to
19 support the descriptions of each entity instance).

20   Each entity class can be referenced not only by its
21 abbreviated name (e.g., EA = "AD") but also by the slot
22 number (e.g., slot .2) where it is stored in the entity
23 definition table 500. The slot number may function as an
24 "entity type number" (abbreviated here as ETN) for
25 numerically identifying its corresponding entity class.
26 Alternatively, an additional "type number" column (not
27 shown) may be added to the ENT.DEF table, 500, unique type
28 numbers may then be entered into each row of the type number
29 column and these can serve as the ETN's. Thus, the
30 "Address" entity class may be referenced not only by the
31 abbreviation EA = "AD" but also by an entity type number
32 whose value, ETN = 2. For the relative table organization
33 (RTO) shown in Fig. 5, the ETN happens to be the same as the
34 slot number (e.g. slot 500.2) where the entity name
35 abbreviation (e.g., AD) is stored in the ENT.DEF table
36 together with the name of the corresponding EiT (e.g., T.
37 Addresses). For the case where an additional type number
38 column (not shown) is added, the unique ETN's can be

1 assigned arbitrarily such as according to the alphabetic

2 ordering of the EA's in which case the ETN's may be used as

3 sort keys for alphabetically ordering the ENT.DEF table rows

4 according to entity class names (e.g. using threaded-list

5 techniques).

6   Referring next to Figure 6, there is shown another

7 table 600 which is also stored within the data storage area

8 130-RP of an engine according to the invention. This table

9 600 may also have a relative-table organization (RTO) and it

10 is referred to as a relations-definition table, or REL.DEF

11 table 600 for short. As before, a matrix notation is used

12 here to identify vertical columns of the REL.DEF table as

13 600a, 600b, 600c, etc.; horizontal rows as 600.1, 600.2,

14 600.3, etc.; and individual boxes as 600a.1, 600a.2, 600b.1,

15 600b.2, etc.

16   The left-most column 600a holds a two character

17 abbreviation representing the class name and/or meaning-

18 string of a relation bubble. The mnemonic, RA, will be used

19 here to designate such a relationship abbreviation. By way

20 of example, box 600a.1 holds the abbreviation "-BU-" which

21 represents the meaning-string "'s Business". (Hyphens

22 embrace the relation abbreviations here to distinguish them

23 from entity abbreviations [EA's].) Box number 600a.2 stores

24 the abbreviation "-OW-" to represent the meaning-string "'s

25 Owning". Box number 600a.3 stores the abbreviation "-SM-"

26 to represent the meaning-string "'s Statement Mailing". Box

27 number 600a.4 holds the abbreviation "-HQ-" to represent the

28 meaning-string "'s Main Headquarters".

29   Each row of the REL.DEF table 600 may also identified

30 numerically by a "relationship type number" (RTN) which in

31 the illustrated example happens to be the same as the slot

32 number (.1, .2, .3, etc.) where its corresponding two

33 character code (-BU-, -OW-, -SM-, etc.) is stored.

34 Alternatively, a type number column (not shown) may be added

35 to the REL.DEF table 600 and unique RTN's may be assigned

36 according to any desired, unique number generating scheme,

37 such as according the alphabetic ordering of the RA's. In

38 the latter case, the RTN's can also function as sort keys

1 for ordering the rows of the REL.DEF table (using threaded
2 list techniques) alphabetically according to relationship
3 class names (RA's). Thus, when given a specific RTN, one
4 can quickly calculate the physical or sequence to the
5 logical address in the REL.DEF table 600 where details about
6 the corresponding relation class are stored so as to quickly
7 retrieve those details.

8   In the second column 600b of the REL.DEF table, there
9 is stored, for each slot (.1, .2, .3, etc.), the name of a
10 single table where instances of the named relation class are
11 stored. The mnemonic, "RiT" (Relation instances Table), is
12 used here to represent such a table. By way of example, the
13 entries in boxes 600b.1, 600b.2, 600b.3 and 600b.4 are
14 respectively: "T.Rel-1", "T.Rel-2", "T.Rel-3" and
15 "T.Rel-1". Note that the entries of box numbers 600b.1 and
16 600b.4 are the same. Compatible instances of two different
17 relation classes may be represented by two corresponding
18 rows of data stored in a common relation-instances holding
19 table (RiT).

20   The third column 600c of the REL.DEF table stores the
21 type number (ETN_h) of a head entity (E-h). Here, the entity
22 type number (ETN_h) is the same as an ETN assigned to a
23 corresponding row in the ENT.DEF table 500 where the
24 abbreviated class name (EA) of that head entity bubble is
25 stored. Similarly, the fourth column 600d of the REL.DEF
26 table stores the type number (ETN_{t1}) of a corresponding
27 first tail entity (E-tl).

28   Note that the first three rows (600.1, 600.2 and 600.3)
29 in Fig. 6A correspond to the relations schema shown in Fig.
30 4A. When row number 600.1 is read across using the column
31 sequence: c, a, d, it corresponds to the relationship
32 descriptor phrase "Customers' business Address". Box 600b.2
33 tells us that instances of this relationship are stored in
34 an RiT table called "T.Rel-1".

35   Similarly, row number 600.2, columns c, a, d correspond
36 to the relationship descriptor phrase "Account's owning
37 Customer". Box 600b.2 tells us that instances of this
38 relation are stored in table T.Rel-2. Row number 600.3

1 likewise corresponds to the relationship describing phrase
2 "Account's statement mailing Address" and tells us that
3 instances of this relation are found in the T.Rel-3 table.
4   The REL.DEF table 600 can be updated indefinitely by
5 adding new rows to its bottom so as to encompass a great
6 number of further relation classes. There is no need to
7 physically order the data describing each of the relational
8 classes and thus descriptions of new classes can be added to
9 the bottom or other empty slots of the REL.DEF table 600
10 sporadically as the need arises over time. Relation classes
11 which become obsolete can be deleted to leave behind an
12 empty slot. Similarly, there is no need to order the entity
13 classes defined by the ENT.DEF table 500. The ENT.DEF table
14 can be updated by arbitrarily adding new entity class
15 describing rows to its bottom or other empty slots or by
16 deleting obsolete entries as the need arises. Accordingly,
17 when demands on the database system of the invention change
18 over time, new relation classes may be defined in
19 combination with new head and tail entity classes. The
20 schema of the invention can be continuously restructured as
21 the need arises simply by updating the REL.DEF and ENT.DEF
22 tables, 600 and 500.
23   The fifth columnar region 600e of Fig. 6A represents a
24 plurality of additional columns within the REL.DEF table
25 600. The names of multiple tail entities which are
26 activated in addition to or in substitution for the first
27 ETN_t of column 600d may be optionally entered in this region
28 600e. Referring briefly to Fig. 6B, an exploded view of
29 this fifth region 600e is illustrated. In the example, each
30 relation class R-x can have as many as five tail entities
31 (T1, T2, T3, T4, T5). The invention is, of course, not
32 limited to five. Column 600d identifies the first tail
33 entity, T1, while extension columns 602 through 605 in
34 region 600e identify the optional, other tail entities,
35 T2-T5. The opening phrase "Customer's business..." of slot
36 number 600.1 columns, c and a, may apply to the first tail
37 entity T1 = "Address" and/or to a second tail entity T2 =
38 "Supplier" and/or to a third tail entity T3 = "Area", etc.

1   Extension region 600e is shown to include a tail
2 activating column 606 which functions as a mask to activate
3 or deactivate each of the corresponding tail entity columns
4 600d, 602-605. In the illustrated example, a dark filled
5 circle means that the corresponding tail entity of that slot
6 (row) is active while an unshaded circle means that the
7 respective tail entity is deactivated. As an alternate
8 embodiment, the mask column 606 may be dispensed with and
9 the lack of an ETN entry (or a "null" entry) in a box of
10 columns 602-605 will be regarded as indicating a deactivated
11 tail while the inclusion of an ETN value will be regarded as
12 indication an active tail. When two or more tail entities
13 are activated, the relation bubble takes on a multi-tailed
14 form such as shown in Fig. 4B at R-4. The same
15 meaning-string is applied to the plural tail entity bubbles
16 of the activated tails. Multiple copies of a prespecified
17 row in the REL.DEF table 600 may be added to empty slots
18 within the table 600 in a boiler-plate stamping manner with
19 only the tail activation masks 606 being modified or some
20 ETN entries of columns 602-605 nulled from copy to copy in
21 order to generate a wide variety of different relation
22 classes.
23   Returning to Fig. 6A, the next column 600f of the
24 REL.DEF table holds a code indicating the cardinality of the
25 corresponding relation bubble (e.g.,{1:m} or {1:1}). The
26 next following column 600g contains a one character code
27 indicating whether there is mandatory coupling (MC) between
28 an instance of the head entity and an instance (or
29 instances) of the tail entity (or active tail entities).
30   Referring to Figure 7 a broader view 700 of a
31 relations-processing storage area 130-RP in accordance with
32 the invention is now shown. Storage means 130-RP is coupled
33 to a data search-and-retrieval machine 815 by way of address
34 bus 131 and data bus 132. Starting at the bottom of Figure
35 7, we see that two relative-organized (RTO) tables are
36 shown: a T.Companies table 710 and T.Addresses table 720.
37 Both of these are Entity-instance Tables (EiT-1 and EiT-2,
38 respectively). The T.Companies table 710 has one column

1   710a in whose numbered slots (710a.1, 710a.2, 710a.3, etc.)
2   are stored the names of various companies. The T.Addresses
3   table 720 has one column 720a in whose slots (720a.1,
4   720a.2, 720a.3, etc.) there are stored data fields
5   representing various street addresses. Each piece of "real"
6   data such as the name of a company (e.g., "Allen's
7   Automobiles") is referred to as an "Entity-instance" or Ei
8   for short. The slot number where the Ei is stored defines
9   an "Entity-instance Number" or EiN for short.
10   The broader view 700 reveals a third table 730 which is
11 labeled in Figure 7 as the T.Rel-1 table and also as RiT
12 730. Each of the numbered slots, 730.1, 730.2, ..., 730.6,
13 etc., in this "Relation-instances Table" (RiT) 730 has five
14 columnar entries. They are respectively: (a) a head
15 entity-type identifier [ETN_h], (b) a head-entity instance
16 identifier [EiN_h], (c) a relationship class identifier
17 [RTN], (d) a first tail entity-type identifier [ETN_t] and
18 (e) a first tail-entity instance identifier [EiN_t]. For the
19 sake of illustrative clarity two-character abbreviation
20 identifiers are shown entered in the vertical columns 730a,
21 730c and 730d of the T.REL-1 table 730. It is within the
22 contemplation of the invention to alternatively enter the
23 corresponding entity or relation type number (ETN or RTN)
24 for these two-character abbreviations. This allows the
25 retrieval machine 815 to quickly and directly access the
26 corresponding row of the ENT.DEF or REL.DEF table where data
27 of interest is stored using either relative-table or key-
28 sequenced access techniques.
29   Columns 730a and 730b in combination identify
30 particular instances of a head entity class (Head Ei) while
31 columns 730d and 730e in combination identify particular
32 instances of a tail entity class (Tail Ei). Referring
33 specifically to box number 730a.2 of the T.REL-1 table 730,
34 the "CU" (or alternatively ETN_h = .1) entry of this box
35 directs the data retrieval machine 815 of the invention to a
36 first section 500.1 of the ENT.DEF table where there is
37 stored the name of a first table (EiT-1 = "T.Companies")
38 where instances of this named entity class ("CU") are

1   stored. The logical link from third table (RiT) 730 to
2   table area 500.a is labeled as L_{731}. The link from table
3   area 500.1 to the first table (EiT-1) 710 is labeled as
4   L_{751}.
5   The second column 730b of the T.REL-1 table holds the
6   slot number or "Entity-instance Number" (EiN = .5 of box
7   730b.2 for example) of the indirectly referenced Entity-
8   instances table (T.Companies 710) within which a specific
9   instance (Ei = "Expert Electronics") of the named head
10 entity class (EA = "CU") is stored. In this example, box
11 number 710a.5 of the first EiT 710 contains the name "Expert
12 Electronics" and this name-string is the entity instance
13 referenced by the "CU.5" entries of boxes 730a.2 and
14 730b.2. The link from box 730b.2 to box 710a.5 is labeled
15 as logical link L_{732}.
16   Referring to columns 730d and 730e of slot number
17 730.2, a similar linkage is created to the instance of a
18 tail entity class. In the illustrated example, the "AD"
19 entry of box 730d.2 points to a second section 500.2 of the
20 ENT.DEF table (thereby defining link L_{733}) where a second
21 pointer is found to a second Entity-instances Table (EiT-2)
22 which in this example is the T.Addresses table 720 (thereby
23 defining link L_{752}). Box 730e.2 holds the slot number (.4)
24 of the indirectly referenced table 720 in which the target
25 data "555 Transistor Lane" is stored (thereby defining link
26 L_{734}). Thus, the illustrated Relationship-instances Table
27 (RiT) 730 defines a connecting relationship (extending from
28 the arrowhead of L_{732} to row 730.2 to the arrowhead of L_{734})
29 which joins the instance "Expert Electronics" of entity
30 class "Customer" (CU) with the instance "555 Transistor
31 Lane" of the "Address" (AD) entity class. Each row of the
32 RiT 730 is referred to as a "Relation-instance" (abbreviated
33 as Ri) and the slot number of that row defines a
34 corresponding "Relation-instance Number" (RiN). (while not
35 shown, it is within the contemplation of the invention to
36 add a "instance number" column to any of tables 710, 720 or
37 730 so as to uniquely identify their rows by arbitrarily
38 assigned instance numbers, EiN or RiN, rather than relying

1   on an RTO slot number, but the RTO slot number approach is

2   believed to result in faster data access.) Columns

3   730a-730b accordingly define the head portion of a

4   "Relation-instance" (Ri) and columns 730d-730e define a tail

5   portion of the relations-instance (as conceptually shown in

6   Fig. 4A). Column 730c, as will now be seen, defines the

7   body portion of each Relation-instance (Ri).

8   Referring to the middle column, 730c, of the T.REL-1

9   table 730, this column holds an identifier pointing to a

10   corresponding row in the REL.DEF table 600 where the

11   relationship class of the instant relationship (Ri) is

12   defined. For the sake of illustrative clarity, the RA of

13   each relation class is shown entered in column 730c. It is

14   within the contemplation of the invention to alternatively

15   enter the corresponding slot number, RTN, of the REL.DEF

16   table 600 so as to speed the access time of the retrieval

17   machine 815. By way of example, the entry "-BU-" in box

18   730c.2 indicates that the relationship between the head

19   instance, Customer.5, and the tail instance, Address.4, is

20   the "'s Business" meaning-string associated with slot 600.1

21   of the REL.DEF table (Fig. 6).

22   The relation instances table, T.REL-1 730, may contain

23   many rows, each of which has the identical head entity-

24   instance entries (in col.s 730a and 730b), identical tail

25   entity-instance entries (in col.s 730d and 730e), but

26   different relationship-defining entries (e.g., -BU-, -HQ-,

27   -OW-, etc.) in column 730c. Each of these almost identical

28   rows would represent a different Relation-instance (Ri). As

29   an example, the address instance AD.4 might be the

30   "Business" address of customer instance CU.5 as shown in

31   slot 730.2. But it may also be the headquarters address

32   "-HQ-" of that same customer CU.5 as shown in slot 730.6.

33   Each of these is considered a different relation instance

34   (Ri). The T.REL-1 table 730 is accordingly shown to include

35   two separate row entries: 730.2 = CU.5-BU-AD.4 and 730.6 =

36   CU.5-HQ-AD.4. A relational query which asks the question,

37   "What is the headquarters address of my customer, Expert

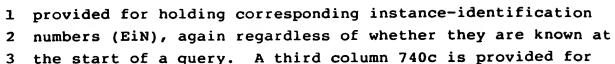38   Electronics?" would be answered by accessing row 730.6 of

1   the T.REL-1 table 730. The slightly different relational
2   query, "What are all the business addresses of my customer,
3   Expert Electronics?" would be answered by accessing all rows
4   in the T.REL-1 table 730 beginning with the entries, "CU.5-
5   BU-", which in the illustrated case includes rows 730.2 and
6   730.5.
7   With the illustrated structuring of a Relation-
8   instances Table (RiT 730), all sorts of relational inquiries
9   can be answered by starting with a known first instance of a
10 first entity class, irrespective of whether the class is a
11 head entity class or tail entity class, and searching
12 through the RiT 730 to locate all relationship-instances
13 (Ri's) of which that starting instance is a member. Once
14 the matching Ri rows are found within a designated Relation-
15 instances Table (RiT), it becomes a simple matter to scan
16 horizontally across the row from the starting instance
17 through the relation descriptor of column 730c to find the
18 corresponding, but until now, unknown instances of the
19 opposed tail and head entity classes.
20   The uncovered instances can then serve as stepping
21 stones for answering further parts of a compound query.
22 Consider for example the two-level query, "What are all the
23 business addresses of my customer Expert Electronics, and
24 once you know that, what other customers use those addresses
25 as their business addresses?" There may be a plurality of
26 business addresses satisfying the first part (Level-1) of
27 the question and each such answer would serve as a new
28 stepping stone leading to the answers which satisfy the
29 second part (Level-2) of the question.
30   In accordance with the invention compound queries are
31 answered by defining one or more question lines in an
32 inquiry-definition (INQ.DEF) table 740. Each question line
33 is identified as belonging to either a one level question or
34 to a particular level of a compound question. A first
35 column 740a of the INQ.DEF table is provided for holding the
36 entity type numbers (ETN) of one or more entity classes,
37 regardless of whether they are known at the start of a
38 query. A second column 740b of the INQ.DEF table is

1   provided for holding corresponding instance-identification

2   numbers (EiN), again regardless of whether they are known at

3   the start of a query. A third column 740c is provided for

4   holding one or more relation type numbers (RTN) while a

5   fourth column 740d is provided for holding corresponding

6   relation-instance numbers (RiN), some of which may be known

7   and others not known at the start of a query. Fifth column

8   740e defines the level of each question row relative to

9   preceding question rows.

10   An RTN value, which if known, is entered in a box of

11 third column 740c in order to indicate to the retrieval

12 machine 815 a corresponding row in the REL.DEF table 600

13 from which the retrieval machine 815 can obtain the name of

14 the single table (RiT-x) where all instances of the named

15 relation type (RTN) reside. The identified table, RiT-x,

16 can then be searched for one or more Ri rows which hold

17 information relevant to a posed query. When found, the RiN

18 values of those rows are entered into one or more boxes of

19 fourth column 740d. The specific Ri rows (e.g., row 730.2)

20 which are fully specified by filled in RTN-RiN data pairs of

21 the INQ.DEF table 740 can then be accessed to direct the

22 retrieval machine 815 to the corresponding head and tail

23 entity instances of interest (e.g., the CU.5 and AD.4);

24 instances which are related to one another by the -BU- entry

25 of box 730c.2).

26   If a specific Ri row is not fully identified at the

27 beginning of a query within a row of the INQ.DEF table 740

28 by a completed RTN-RiN pair, the Ri row or rows of interest

29 can be nonetheless located by partially filling in a row

30 within the INQ.DEF table 740 and then searching the REL.DEF

31 or ENT.DEF tables for additional information. Row 740.2 of

32 the INQ.DEF table is shown to have the question line,

33 "??.?-HQ-?" which may mean "Please identify the Headquarter

34 addresses of all my customers". In such a case, all rows of

35 the T.REL-1 table 730 which have the entry -HQ- in their

36 middle column 730c would provide the required information.

37 Each such -HQ- row of RiT 730 would pair an identified

38 instance of a Customer (head Ei) with an identified instance

1 of a headquarters Address (Tail(1) Ei). It is to be

2 appreciated that for cases of multi-tailed relation classes,

3 the corresponding RiT would have columns for identifying the

4 other tail entity instances (e.g. Tail(2) Ei, Tail(3) Ei,

5 etc., not shown).

6   Sometimes a question is more specific. By way of

7 example, let it be assumed that an inquiring user has a

8 specific but fragmentary piece of starting information such

9 as the street address "555 Transistor Lane". The inquiring

10 user wishes to find out the names of one or more companies

11 for whom "555 Transistor Lane" is a "Business Address". The

12 user identifies the fragmentary information to the data

13 retrieval machine 815 as belonging to the "Address" entity

14 class. In response, the machine 815 searches through the

15 ENT.DEF table 500 to locate the entity type number "ETN" of

16 the named class and the Entity-instances Table "EiT" where

17 all instances of this "Address" entity class are stored.

18 It should be recalled that the illustrated relative-table

19 organization "RTO" of the ENT.DEF table 500 is not mutually

20 exclusive of a key-sequenced organization "KSO". According

21 to the invention, the EA column 500a of the ENT.DEF table is

22 threaded alphabetically so that the row of a desired entity

23 class (e.g., EA = "AD") can be easily found using known key-

24 sequenced search algorithms. A different table (not shown)

25 can serve as an abbreviation to full name look-up table for

26 converting between the entity name abbreviation (EA) and the

27 full name or narrative description of the entity class (ECN)

28 if desired or, alternatively, the ENT.DEF table 500 may

29 include one or more additional columns (not shown) for

30 providing this search and conversion function.

31   Once the corresponding type number (ETN) of the entity

32 class is identified, in this case ETN=.2 referencing slot

33 500.2, the retrieval machine 815 places this first puzzle

34 piece into an appropriate box of the INQ.DEF table. In this

35 example it will be box 740a.3 of INQ.DEF question line 740.3

36 which is for illustrative purposes filled with the

37 corresponding EA="AD".

38   The retrieval machine 815 then obtains from box 500b.2

1   of the ENT.DEF table the name of the corresponding EiT where

2   it is to search for the occurrence of the fragmentary

3   information "555 Transistor Lane". The EiT's can be key-

4   sequence organized (KSO) in addition to their RTO

5   structuring to facilitate such searching. After the

6   corresponding EiT (in this case, the T.Addresses table 720)

7   is searched and the row of the fragmentary information is

8   found, its corresponding EiN, in this case .4, is entered as

9   an entity-instance number (EiN) in box 740b.3 of the INQ.DEF

10 table 740.

11   The earlier found entity type number (ETN) which

12 corresponds to EA = "AD" now combines with the EiN = .4 of

13 INQ.DEF row 740.3 to define the "starting instance" for

14 resolving question line 740.3. The starting instance is

15 AD.4.

16   The relationship type number (RTN) of the relationship

17 under question (-BU-) is entered in box 740c.3. If the RTN

18 value is not known, the REL.DEF table 600 is first searched

19 to generate the appropriate RTN. While not shown, the

20 REL.DEF table or some other table will include a full name

21 or narrative column for converting between a relationship's

22 full name/description and its abbreviated form (RA). Box

23 740d.3 is now the last puzzle piece to be filled in as

24 indicated by a question mark in Fig. 7.

25   Since the ETN.EiN-RTN- entries of boxes 740a.3, 740b.3

26 and 740c.3 are now all known, the retrieval machine 815

27 searches through the corresponding RiT (T.REL-1 table 730)

28 to locate all relation-instances (Ri's) which have the

29 corresponding ETN plus Ein in the tail entity instances

30 columns 730d and 730e and the corresponding RTN in column

31 730c. The REL.DEF table 600 identifies the starting entity

32 class of the AD.4-BU-? question as being a tail entity.

33 (When there is more than one tail entity, the RiT will have

34 plural columns for identifying first, second, etc. tail

35 instances and the REL.DEF table 600 will specify which of

36 these tail columns is to be searched.) In the illustrated

37 example, row 730.2 of the T.REL-1 table will be found to

38 have matching information. The retrieval machine 815 can

1 now fill the last empty box 740d.3 of the INQ.DEF row 740.3

2 with the information RiN = .2. Once question row 740.3 is

3 completely filled, the retrieval machine 815 may use the

4 information of this INQ.DEF row 740.3 to retrieve the

5 detailed information about the head entity instance,

6 Ei = "Expert Electronics" from table row 710a.5 of the

7 T.Companies table.

8   The ETN.EiN identifiers of the uncovered Level-1

9 answer, "Expert Electronics" can now serve as stepping

10 stones which fuel a second part of a compound query. For

11 example, the full query might have been "Who has business

12 address, 555 Transistor Lane and what bank accounts belong

13 to the entity or entities that satisfy the first part of

14 this question?" The first part is defined here as "Level-1"

15 of the question and the second part as "Level-2". Column

16 740e of the INQ.DEF table is shown to identify the level

17 number. Referring to a feedback link L_{744} shown in Fig. 7,

18 the Level-1 answer (ECN = "CU" and EiN = .5) can now be fed

19 back as an entry to a subsequent inquiry-defining row 740.4

20 so that the multi-level inquiry path may continue. Inquiry

21 box 740c.4 is shown already filled with the relationship

22 identifier (-OW-) for locating account owners. The answer

23 to inquiry row 740.4 may be used to fuel yet a further level

24 (Level-3, not shown) of a compound inquiry and the answer or

25 answers to that inquiry may fuel yet further inquiry rows.

26   Referring to Fig. 8, a block diagram of a database

27 system 800 in accordance with the invention is shown. Bulk

28 storage means 130* is indicated to include a relation-

29 processing region 130-RP in accordance with the invention.

30 The bulk storage means 130* may also include previously-

31 utilized relational tables for defining "implied" relation-

32 ships between entities. Such "implied" relationships are

33 not incompatible with the "explicit" relationships that are

34 defined by the REL.DEF table 600 of the invention. As shown

35 in region 130-RP of Fig. 8, the REL.DEF table and ENT.DEF

36 table may be used to define a continuously expandable

37 backbone which supports various relationships (RiT-1, RiT-2,

38 etc.) between various entity instances (EiT-1, EiT-2, EiT-3,

1   etc.). The INQ.DEF table may be visualized as having two
2   legs (dashed vertical lines) which sequentially step from a
3   starting instance table (EiT-1), across a starting table of
4   relationship instances (RiT-1) to an explicitly linked table
5   which holds relationship-opposed instances (EiT-2) of the
6   starting instances. The opposing instances (of EiT-2) then
7   become starting instances for a next inquiry step over yet a
8   further set of relationship instances (RiT-2).

9   Since the REL.DEF and ENT.DEF tables may be expanded as
10 desired by adding new entries to empty middle or bottom
11 slots found within them, a lay user can create new entities,
12 new relation classes and restructure the schema of
13 explicitly-defined relationships and entities forever
14 without having to reprogram the database system 800 at the
15 source or object code level. Instead, the lay user supplies
16 schema restructuring commands, in an appropriate structured
17 language, as indicated at 870 for restructuring the schema
18 whenever needed. The access control program 820d of the
19 retrieval machine 815 may remain fixed while the entity-to-
20 explicit-relationship schema of region 130-RP is forever
21 changed. Accordingly, object-code compilation 814 needs to
22 occur only once. The source code listing 812 of this access
23 control program needs to be developed and debugged only
24 once. Substantial cost savings are realized, especially as
25 time progresses and new entity-relationship schemas are
26 required.

27   In some commercial applications, the ENT.DEF table and
28 REL.DEF table may be relatively short, having for example
29 less than 1000 rows each (e.g., the ENT.DEF table may have
30 30 rows or less and the REL.DEF table may have approximately
31 100 rows or less). For suchy cases it has been found
32 advantageous to "copy" the ENT.DEF and REL.DEF tables from
33 the bulk storage means 130* to a higher speed memory area
34 within first memory means 120 in order to shorten processing
35 time. The copied versions of the ENT.DEF and REL.DEF tables
36 can be purely-key-sequenced if an additional "type number"
37 column is added for storing the respective ETN's and RTN's
38 of each row. The higher data access speed of the first

1 memory means 120 more than compensates for any speed

2 reduction which might be caused by switching to a purely

3 key-sequenced organization. These "mirror" copies of the

4 ENT.DEF and REL.DEF tables are then accessed by the CPU 110

5 in place of the original ENT.DEF and REL.DEF tables. It is

6 advisable to periodically check the original ENT.DEF and

7 REL.DEF tables for possible revisions, since lay users may

8 update that original tables at any time, and when such

9 revisions are detected, to immediately recopy the ENT.DEF

10 and REL.DEF tables into the first memory means 120 so that

11 the mirror tables faithfully reproduce the contents of the

12 original tables.

13   The CPU 110 in combination with the various modules of

14 the object code 820d can be visualized as one or more

15 machine means for performing data-altering functions as

16 specified by the object code 820d. A Microfiche Appendix is

17 included here listing sample modules written in Tandem

18 COBOL'85™ and TANDEM SCREEN COBOL™ for execution on a Tandem

19 NONSTOP™ computer system running under Tandem NonSTOP SQL™,

20 TMF™, Pathway™, SCOBOLX™ and Guardian™ systems (all

21 available from Tandem Computers of Cupertino, California).

22 It is to be understood that the sample modules disclosed in

23 the Microfiche Appendix are merely exemplary. The invention

24 may be practiced using different computer hardware and/or

25 software.

26   Referring to Fig. 9, a schematic diagram of an inquiry

27 processing engine 900 in accordance with the invention is

28 shown. The engine 900 comprises an inquiry guide means 910

29 which is coupled to a relationship defining means 960, a

30 relationship storage and search means 970 and to an

31 intermediate-answers receiving means 980. The intermediate

32 answers means 980 feeds abbreviated answers back to the

33 inquiry guide means 910 after such answers are produced by

34 the relation storage means 970. Desired ones of all

35 produced results are sent from the inquiry guide means 910

36 to an abbreviated results gathering means 915 which then

37 expands them into full result details by sending an entity

38 type signal $sETN_x$ to an Entity Define means 950 which

1 includes within itself, the earlier described ENT.DEF table

2 500. The sETN$_x$ signal is converted by the entity define

3 means 950 into an entity table selecting signal sEiT which

4 is fed into an entity storage means 920 that includes within

5 itself a plurality of entity-instances tables (EiT-1, EiT-2,

6 etc.) such as earlier described. Results gathering means

7 915 also feeds an instance row selecting signal, sEiN, to

8 entity storage means 920. Details from the addressed entity

9 instance row are then transmitted through a details filter

10 985 and portions of the details which are selected by the

11 filter 985 are then printed on a detailed results display

12 (e.g. a video monitor) 990.

13   Relationship inquiry in general is a two step

14 operation: path selection (to create an Inquiry) and

15 inquiry execution. On a Path Selection screen (not shown)

16 the operator selects starting and optionally ending entity

17 types and supplies detailed description of the path to

18 follow. Each path is defined in terms of:

19   a starting entity type to initiate the query path,

20   a connecting relationship type which will lead to

21   an intermediate entity type and then to another

22   connecting relationship type and another intermediate

23   entity type, and so forth until

24   . . .

25   a last connecting relationship type leads to a

26   terminating entity type

27   Taking out all but the key words from the above, we get

28 the form structure:

29   <starting entity type>

30   <connecting relationship type> <intermediate

31   entity type>

32   <connecting relationship type> <intermediate

33   entity type>

34   . . .

35   <connecting relationship type> <terminating entity

36   type>

37   A single inquiry definition may initiate several

38 parallel paths which extend from a starting entity type to

1   an ending entity type. When the ending entity type has not

2   been specified in the header of the path-selecting screen

3   then all these parallel paths can end with different entity

4   types. For example, an inquiry to show a person's total

5   involvement with all accounts held at a bank could be

6   defined as shown in the following Table I:

7

8   TABLE I

9

Entity	Level-1 Relationship	Connected Entity	Level-2 Relationship	Connected Entity
Person	--->Account Holder	--->Account		
Person	--->Loan Guarantor	--->Account		
Person	--->Signatory	--->Account		
Person	--->Card Holder	--->Account		
Person	--->Group Member	-->Joint Party	--->Account Holder	--->Account
Person	--->Group Member	-->Joint Party	--->Card Holder	-->Account

        Each of the above lines is a separate path generated by

one inquiry form. The results of the inquiry would show all

Accounts a Person had influence over, either directly or as

a member of a partnership.

        For simplicity the above inquiry is shown on the screen

as in the following Table II:

                            TABLE II

Level	Relationship	Entity
1	Account Holder	Account
1	Loan Guarantor	Account
1	Signatory	Account
1	Card Holder	Card
1	Group Member	Joint Party
2	Account Holder	Account
2	Card Holder	Card

1   Note that level numbers are used to determine which
2 entity types are intermediate to a path, which entity types
3 terminate a path, and which relationship types commence a
4 new parallel path. A line containing a level number which
5 is the same as that of an immediately previous line
6 indicates a parallel path separate from the previous line.
7 A level number greater than that on the previous line
8 indicates the entity on the previous line is an intermediate
9 entity, (i.e. the path is an association, and will follow
10 several relationship links before terminating the path.)
11   Once a set of paths have been stored as an inquiry and
12 recorded in the system it may be executed. Each unique set
13 of inquiries is given a unique name, stored as such in the
14 inquiry-definition table (INQ.DEF) and may be recalled for
15 execution repeatedly at any time without need to go through
16 the path selection process again. Before executing the pre-
17 defined inquiry, the operator must select one or more
18 starting entity instances for which the query is to run.
19 Hence for each execution of an inquiry, the operator must
20 choose which occurrence of the Starting Entity Type to
21 use. Using the previous sample inquiry to investigate
22 persons of the names, "John Smith" and "Bill Brown", the
23 operator would execute the same inquiry once using "John
24 Smith" as the Starting Entity instance and once using "Bill
25 Brown" as the Starting Entity instance.
26   The Inquiry is executed by examining each of the
27 defined paths in turn. Starting with the selected entity
28 and following the first relationship, a list of intermediate
29 (or target) entities is assembled. For each of the
30 intermediate entities the next leg of the path is followed
31 through the level 2 relationship etc. until the inquiry
32 operation arrives at the ending entity type at which time
33 the results of the entire path (with all intermediate
34 entities and relationships) may be displayed to the
35 operator.
36   If the ending entity type has been specified during
37 inquiry definition, then at execution time the operator may
38 select not only the starting entity occurrence of interest

1 but also the occurrence of an ending entity. In this case
2 the inquiry will return results from only the paths that
3 satisfy this termination condition.
4   Reusable inquiry sets would normally only be created by
5 privileged users. However, each inquiry set that is created
6 for subsequent executions may be given its own security
7 settings and attached to its own menu. Hence where
8 sensitive data was involved, normal operators would be given
9 access to only those inquiry sets they specifically need for
10 their day to day business operations.
11   Despite its complexity, the inquiry engine 900 of the
12 invention can operate at high speed because the EiT and RiT
13 structures, while they may be large in size rely on relative
14 tables. Relative table structures have always offered high
15 performance for Random memory access (as opposed to key-
16 sequenced access) but presented many complications and
17 difficulties in other areas of use (e.g. updating). Because
18 of this, conventional wisdom has been to use purely
19 Key-Sequenced structures almost exclusively. Key-Sequenced
20 structures pay performance penalties for the use of extra
21 indexing levels.
22   The first problem with Relative structures was that
23 with some early versions, deleted row locations (or slots)
24 could not be re-used without file (table) reorganization.
25 Reorganization of Relative structures in this case meant
26 compressing the file (table) to regain unused slots. This
27 process can change the relative addresses from their
28 original values, which can cause corruption of the
29 database. Reorganization is no longer required because
30 Relative structures such as offered in Tandem's NonSTOP SQL™
31 system allow deleted row slots to be reused immediately.
32 The Tandem system actually ensures that vacated slots are
33 used again and again. Relative tables in NonSTOP SQL™ can
34 be partitioned and re-partitioned without risk of corrupting
35 the database, but table compression is no longer necessary
36 or allowed. Partitioning a table means that the table can
37 be split across a plurality of data storage devices, usually
38 disks, transparent to the object code of the

1 application/program running under NonSTOP SQL™.

2   The second problem with Relative structures was
3 implementing meaningful keys that allowed access to the data
4 in a sequence based on indicative data, such as numerical
5 order of account number or alphabetical order of customer
6 name.   However, by using Alternate Key index tables it is
7 possible to provide meaningful sequential access of entities
8 stored within Relative Tables.

9   The Relationships Processor or "engine" of the present
10 invention is a "Closed Loop" system in that all explicit
11 schema definitions are stored within the system.   The finite
12 set of tables and their meanings are also defined within the
13 system.   This provides an infrastructure that makes the
14 Table Structures transparent to users and developers.
15 Hence, Relative tables can be used for performance
16 improvements while avoiding any usability penalties that
17 once existed.

18   Hence this invention has gone against conventional
19 attitudes because of new data processing techniques used by
20 the invention.

21   The above advances in Relative structure techniques,
22 coupled with the closed loop nature of the Relationships
23 Processor has allowed Relative tables to be used in a
24 controlled and meaningful way, destroying the premise that
25 Key-Sequenced structures are the best way to store
26 relationships.

27   A benchmark was run on a Tandem NonSTOP SQL™ system to
28 test the system's performance capabilities.   The benchmark
29 was to simulate the normal processing requirements of an
30 extremely large bank's Customer Information System.

31   The database used 14 Gigabytes of disk storage space,
32 and was populated with 5 million Customers, 7 million Cards,
33 9 million Addresses, 10 million Accounts and 67 million
34 relationships.

35   The benchmark simulated 1000 simultaneous users
36 (tellers), with each user executing 100 typical on-line
37 transactions.

38   The invented system achieved a rate of 64 transactions

1 per second with less than 2.6 second response time for 90%
2 of all transactions which included all screen formatting.
3 This is quite remarkable for a database system of this size
4 and complexity.
5   The invented system was also benchmarked for batch
6 processing at rates of hundreds of transactions per
7 second. This shows that the system is able to process
8 inquiries at commercially acceptable rates.
9   Referring to Fig. 9, an inquiry begins by transmitting
10 a signal representing starting entity instance and relation
11 information (e.g., "Level-1 = $ETN_1 \cdot EiN_1 - RTN_1 - ?$") from an
12 input form means 901 to the inquiry guide means 910. The
13 data of this starting instances and relationship signal,
14 902, is stored in an inquiry-defining table 740 provided
15 within the inquiry guide means 910. The inquiry guide means
16 910 transmits a starting relationship type signal $sRTN_1$ to
17 the relation defining means 960 and a relationship instance
18 defining signal $sRi = ETN_1$ and/or EiN_1 and/or RTN_1 to the
19 relationship storage and search means 970. The relation
20 defining means 960, which includes REL.DEF table 600,
21 transmits a Relation-instances table selecting signal $sRiT_1$
22 to the relationship storage means 970 in order to select one
23 of a plurality of Relation-instances tables, RiT_1, RiT_2,
24 RiT_3, etc. stored within the relation storage means 970.
25 The relation defining means 960 further transmits a head
26 or tail identifying signal, H/T, to the relation storage
27 means 970 to identify a head or tail instance defining
28 column, Ei-h or Ei-t, which should be searched for
29 information matching the ETN_1 and/or EiN_1 information of the
30 starting instance signal, sRi. (While not shown, each RiT
31 can have multiple columns specifying a plurality of tail
32 entity instances, i.e., Ei-t1, Ei-t2, etc. and in such a
33 case, the H/T signal also indicates which one or more tail
34 columns of the target RiT are to be searched for matching
35 information.) In response, the relationship storage and
36 search means 970 searches through the selected relationship
37 instances table RiT-x to find information matching that of
38 the input signals, sRi, sRiT and H/T. Signals 971

1   representing the opposing entity instances (Ei-o) of each

2   matched row are then transmitted to an intermediate answer

3   gathering means 980 which compiles within its memory area a

4   list of entity instances, $Ei-o_1$, $Ei-o_2$, $Ei-o_3$, etc., which

5   oppose the starting entity instances found in matching rows

6   of the referenced RiT (730). The collected intermediate

7   answers are then fed back along path 981 to the inquiry

8   guide means 910 in order to fill stepping-stone boxes (shown

9   as still open question, ?.??) in a next level query row

10 (e.g. Lvl-2). The next query row (e.g. Lvl-2) now becomes

11 the new starting row and its contained information, Ei-

12 $o_2-RTN_2-?$, is now fed as the new sRi signal to the relation

13 storage means 970 and the relation define means 960. The

14 inquiry loop repeats until an inquiry path terminates on its

15 own or a terminating entity is struck.

16   After termination, the results of the inquiry loop are

17 fed through signal bus 911 to an abbreviated results

18 compiling means 915 which orders the results according to

19 their level number and interrelation. By way of example, a

20 first Level-2 inquiry may produce intermediate answer,

21 Ei-2a. That intermediate answer together with its forward-

22 connecting relation (RTN_2) may produce a plurality of

23 intermediate answers at Level-3, namely, Ei-32a.1, Ei-32a.2,

24 etc. Each of these Level-3 answers may then result in a

25 larger plurality of Level-4 answers (not shown) and so

26 forth. Likewise the Level-2 answer Ei-2b may produce a

27 plurality of Level-3 answers, Ei-32b.1, Ei-32b.2, Ei-32b.3,

28 etc. Each of these answers is recorded as a paired set of

29 an entity class number ETN and an entity instance number

30 EiN. The abbreviated results are then expanded into

31 user-understandable results by sending an entity type number

32 signal, $sETN_x$ to the entity definition means 950 and a

33 corresponding entity instance signal, sEiN to the entity

34 storage means 920. In response the entity storage means 920

35 then produces detailed information from the referenced

36 entity instances tables. Often, the database user may not

37 wish to see all of the detailed information within a row,

38 but rather wishes to see only prespecified columns of the

1 referenced row and wishes the data to be displayed according

2 to a predetermined display format. The details filter 985

3 filters out information from undesired columns and orders

4 the remaining data according to a predetermined display

5 format selected by the user. The desired "real" information

6 then appears in the selected format on display means 990.

7   Referring to Fig. 10, it will now be explained how a

8 single starting instance can lead to the production of a

9 large plurality of answers. A database user has a first

10 account number (instance $I_{a/E1}$) from which the user wishes

11 to find all persons, groups or companies which are holders

12 of that account, and once known, all other accounts held by

13 those persons, groups or companies; and further, where a

14 person is a member of a group or a group has many persons as

15 its members or where a company has subsidiary companies, the

16 accounts held by these entities. As shown in Fig. 10, the

17 relationship instance $I_{a/R1}$, has three tails, T1, T2 and T3,

18 only one of which will be active for a given instance of the

19 head entity $I_{a/E1}$. Tail T1 points to person instance $I_{b/E2}$.

20 Tail T2 points to group instance $I_{b/E3}$. Tail T3 points to

21 company instance $I_{b/E4}$. These instances of person, group

22 and company represent intermediate instances which lead to

23 the desired answer, namely, the accounts held by such

24 persons. One person $I_{b/E2}$, may hold many other accounts as

25 indicated by the multiple instances of the 's Holder

26 relationship instances, $I_{i/R1}$, $I_{j/R1}$, $I_{k/R1}$, etc. Each of

27 these relationship instances has a corresponding account

28 instance at its head (H) end. In Fig. 10, these are $I_{i/E1}$,

29 $I_{j/E1}$, $I_{k/E1}$, etc. The rest of Fig. 10 is self-

30 explanatory. A person can belong to several groups and each

31 of those groups may hold several accounts. A group may have

32 many members and each of those members may have several

33 accounts. A company may be a subsidiary of many other

34 companies and each of those companies can hold several

35 accounts. Thus, the list of ending instances shown in

36 Fig. 10, $I_{i,j,k/E1}-I_{x,y,z/E1}$, can be quite long compared to

37 the starting instance $I_{a/E1}$ which started the inquiry.

38   A variety of modifications will become apparent to

1 those skilled in the art in light of the above description.

2 The scope of the claimed invention is accordingly, defined,

3 not by any specific embodiment described herein, but rather

4 by the following claims.

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38